Approximate Algorithms for Document Placement in Distributed Web Servers

Savio S.H. Tse

Abstract—We study approximate algorithms for placing a set of documents into M distributed Web servers in this paper. We define the load of a server to be the summation of loads induced by all documents stored. The size of a server is defined in a similar manner. We propose five algorithms. Algorithm 1 balances the loads and sizes of the servers by limiting the loads to k_l and the sizes to k_s times their optimal values, where $\frac{1}{k_l-1}+\frac{1}{k_s-1}\leq 1$. This result improves the bounds on load and size of servers in [10]. Algorithm 2 further reduces the load bound on each server by using partial document replication, and Algorithm 3 by sorting. Algorithm 4 employs both partial replication and sorting. Last, without using sorting and replication, we give Algorithm 5 for the dynamic placement at the cost of a factor $O(\log M)$ in the time-complexity.

Index Terms—Distributed Web server, load balancing, document placement, document replication, file allocation problem, approximate algorithm, NP-completeness.

1 Introduction

There has been a tremendous demand of high performance internet Web servers for many years. Standalone Web servers are often unable to provide reliable and scalable services. In contrast, distributed solutions are a natural and popular way to improve the reliability, scalability, efficiency, and availability of a system. The advantages of distributed solutions in a cluster are higher computing power and fault-tolerance with graceful degradation. Their disadvantages, however, are the complexity involved in scheduling and balancing the work of individual servers. In this paper, we address the problem of balancing load and required storage space among all servers under different assumptions on the input documents.

Many papers have addressed the load balancing problem [5], [10], [11], [15], [17]. Their solutions can be classified into three categories: the dispatcher-based approach, the domain-name-server-based (DNS-based) approach, and the (context-aware) document placement approach. One can refer to [6] for a survey of the variants of the dispatcher-based approach; and one can refer to [4], [7], [8], [11], [12] for the DNS-based approach. These two approaches assume full or excessive replication of documents in the distributed servers. However, excessive replications induce much overhead for data transfers and replica updates, especially when the servers are hosted in a network with limited bandwidth or the number of documents is large. Along with this, the cost of storage space is also high, of course, in a sense that the spaces are not efficiently used. The inefficiency of space usage confines the related algorithms to storing the replicas in on-disk caches, rather than in memory. This space-cost will in turn affect the performance of Web server systems.

In this paper, we consider the document placement approach. This approach uses a kind of algorithms called *document placement algorithm* to store documents in servers. One of the purposes of this kind of algorithms is to balance the workload among the servers. A static document placement algorithm determines the mapping from each document to a server,[1] and allows another (phase of the) algorithm to place the documents into the servers. A dynamic (or online) version computes and places one document at a time. In Section 3.5, we develop algorithms for dynamic placement in which the execution time slows down by a factor $\log M$, compared with static placement. Since it is not supposed to do a full or excessive replication, the dispatcher must be *content-aware* [18] in order to forward each client request to the precise server. The price of the content-awareness is the high overhead associated with looking up the content of a request in the dispatcher. It should be noted that the load balancing problem might not be resolved by using a single approach out of the above three, but may require a hybrid of them.

The document placement problem is a variant of the classical *File Allocation Problem* (FAP). In [9], Ceri et al. gave a solution for the optimal FAP based on the classical Knapsack Problem. Fisher and Hochbaum studied the trade off between the cost of assessing data and storing replicated copies [14]. A survey of results of FAP before 1982 can be found in [13]. In 1996, Simha et al. extended the idea of FAP to parallel Web servers [19]. Many of these previous works used heuristics; however, none of them focused on the trade off between the bounds of the loads and spaces of file servers. In [20], Zhou et al. gave algorithms for balancing the loads empirically, but not in theory. In [17], Narendran et al. proposed the Binning Algorithm which allows a document to be replicated to multiple servers. Different replicas may have different probabilities of being accessed. The Binning Algorithm balances the loads optimally, but the system needs counters to count the frequencies of client requests received. Moreover, this algorithm does not balance the space utilization, and the space-inefficiency will increase time-cost as mentioned. In [10], Chen and Choi showed the optimal placement problem can be reduced to the bin packing problem in polynomial time. Thus, the

• *The author is with the Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong. E-mail: sshtse@cs.hku.hk.*

Manuscript received 15 Jan. 2004; revised 30 June 2004; accepted 13 Sept. 2004; published online 21 Apr. 2005.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0018-0104.

1. The algorithm maps a document to a subset of servers if document replication is allowed.

placement problem is NP-hard and for this reason two approximation algorithms have been proposed. The first algorithm results in a factor 2 of the optimal load distribution (per HTTP connection). The second algorithm results in a factor 4 of the optimal load distribution using at most four times the optimal storage space. In this paper, we first improve Chen et al.'s result by giving a generalized solution for the load-space trade off in Theorem 1. From this theorem, one could choose a better load balancing performance at the expense of space utilization. On the other hand, we could balance the spaces of the distributed servers at the expense of load distribution. Extending this result, we improve the result for a specific case in which the input documents are sorted according to their loads. We also improve the load-space balancing performance by replicating documents where the input is both sorted and unsorted. Last, we consider the dynamic aspects of the problem. In our model, we assume each replica of a document shares the same probability of being assessed.

2 Definitions and Models

In our model, we have M homogeneous servers with capacity C, and N independent documents. As a common phenomenon, we assume $N \geq M$. For all $i \in [1, N]$, the ith document has a positive size s_i and load l_i. The load of a document is the product of its access rate and size. We assume $s_i \leq C$ for any $i \in [1, N]$, and $\sum_{i\in[1,N]} s_i \leq MC$. The load of a server is the sum of loads of all documents allocated to it. A similar definition applies for the size of a server. The size of a server cannot exceed C, while the load of a server can be unbounded. Let L and S be the optimal bounds on the load and size of each server, respectively, where

$$L = \max\left(\max_{i\in[1,N]} l_i, \frac{\sum_{i\in[1,N]} l_i}{M}\right), \text{ and}$$
$$S = \max\left(\max_{i\in[1,N]} s_i, \frac{\sum_{i\in[1,N]} s_i}{M}\right).$$

Let $k_l > 2$ and $k_s > 2$ be two numbers satisfying the *fundamental inequality*

$$\frac{1}{k_l - 1} + \frac{1}{k_s - 1} \leq 1. \quad (1)$$

These two numbers play important roles in the upper bounds of load and size. The fundamental inequality is to guarantee the existence of a server with load and size bounded by $(k_l - 1)L$ and $(k_s - 1)S$, respectively, before we place a document. The motivation of this inequality will become clear in the proof of Theorem 1.

If a document is replicated in another server, its replicas will share the load equally. Each replica will be of the same size as the original document. For all $i \in [1, N]$, let r_i be the number of replicas of the ith document, including its original copy. Precisely, $\forall i \in [1, N]$, if the ith document has replicas, each of its replicas will be of a size s_i and a load $\frac{l_i}{r_i}$. (In [17], different replicas may have different loads.) It is also reasonable to place all replicas of the same document in different servers. Hereafter, by saying that a document has r replicas, we mean there are r copies of the document including the original one. For the case that we do not replicate the ith document, it still has (only) one replica which is itself, i.e., $r_i = 1$.

It is already well-known that the general problems associated with the allocation of documents so as to bound the loads or sizes of all servers are NP-hard [10]. The bin-packing problem can be reduced to some special cases of these problems. Using the same argument, the problems associated with bounding the number of replicas for each document so as to achieve the optimal load for all servers is also NP-hard.

In the following sections, we find P-Time approximate algorithms for placing the documents. The price for the improvement in time complexity is to release the optimality of the bounds. In Section 3.1, we try the simple greedy method for bounding the loads and sizes of all servers to $k_l L$ and $k_s S$, respectively. In Sections 3.2, 3.3, and 3.4, we apply the sorting and/or replication to achieve better solutions.

3 Approximate Algorithms and Their Analyses

For all algorithms discussed in this section, the inputs are N documents with individual loads and sizes, and M servers, which are initially empty sets of documents and replicas. The output is the M servers with documents and/or replicas allocated. We assume each server keeps the knowledge of its last input document. The function $Last(X)$ outputs the latest document placed in server X. Note that replication of documents will not increase the total load of all the servers, but will increase the total size. Therefore, in our discussion, the total load is always $\sum_{i\in[1,N]} l_i$.

3.1 Algorithm 1—Simple Placement

We first give Algorithm 1 without applying sorting and replication. This is the simplest document placement algorithm in this paper. We relate the parameters k_l and k_s to the upper bounds on load and size of each server in Theorem 1.

Algorithm 1 (k_l, k_s):
1. Compute L and S;
2. $SERVER :=$ set of all servers;
3. For every document d
3.1 Find a server $X \in SERVER$;
3.2 Place d into X;
3.3 If (the load of X is at least $(k_l - 1)L$) or (the size of X is at least $(k_s - 1)S$)
$SERVER := SERVER - \{X\}$;

Theorem 1. *After execution of Algorithm 1(k_l, k_s), all documents can be placed such that each server's load is less than $k_l L$ and size less than $k_s S$.*

Proof. Consider a document d. Before placing d on any server, there are less than $\frac{1}{k_l - 1} M$ servers which loads are at least $(k_l - 1)L$; otherwise, total load is greater than ML, where $ML \geq \sum_{i\in[1,N]} l_i$. A contradiction.

On the other hand, there are less than $\frac{1}{k_s - 1} M$ servers which sizes are at least $(k_s - 1)S$; otherwise, total size is greater than MS, where $MS \geq \sum_{i\in[1,N]} s_i$. A contradiction.

Hence, by the fundamental inequality, there exists at least one server X which load is less than $(k_l - 1)L$ and size is less than $(k_s - 1)S$. After placing d into X, X's load is less than $k_l L$ and size is less than $k_s S$. Inductively, we prove the lemma. □

In particular, if we assign $\frac{5}{2}$ to k_l and 4 to k_s, we will improve the result by [10], the load and size bounds of which are both 4.

The data structure of $SERVER$ can be a (cyclic) link list. Therefore, inside the for-loop, each step needs $O(1)$ time. Step 1 will take $O(N)$ time and Step 2 $O(M)$. Hence, the time-complexity for Algorithm 1 is then $O(N)$.

3.2 Algorithm 2—Placement with Replication

In this section, we propose a partial replication technique for replicating some of the documents and balancing the loads of servers. We call this technique *selective replication* because we will replicate only the top documents of those servers having loads higher than our load bound $(k_l - \frac{1}{2})L$. We replicate a document by making one more replica. This is because we do not want to place a heavy burden on the size bound.

Replicating documents will not increase the total load, but will inevitably increase the total size. However, if we make only one extra copy for each replication, Theorem 2 shows that the load bound can be decreased by $\frac{L}{2}$ without increasing the size bound. Therefore, in the proof of Theorem 2, we try to avoid the effect of increasing total size.

Algorithm 2 (k_l, k_s):
1. Perform Algorithm 1(k_l, k_s);
2. Let $SERVER$ be the set of all servers which loads are less than $(k_l - 1)L$ and sizes are less than $(k_s - 1)S$;
3. For every server X with a load at least $(k_l - \frac{1}{2})L$
3.1 $d := Last(X)$;
3.2 Find a server $Y \in SERVER$;
3.3 Replicate d and put the replica into Y;
3.4 If (the load of Y is at least $(k_l - 1)L$) or (the size is at least $(k_s - 1)S$)
$SERVER := SERVER - \{Y\}$;

Theorem 2. *After execution of Algorithm 2(k_l, k_s), all documents are placed such that each server's load is less than $(k_l - \frac{1}{2})L$ and size less than k_sS.*

Proof. After execution of Algorithm 1(k_l, k_s), if the loads of all servers are less than $(k_l - \frac{1}{2})L$, then it is done. We now assume the other case.

Let P be the set of servers which loads are at least $(k_l - \frac{1}{2})L$, A be the set of servers which loads are at least $(k_l - 1)L$. $P \neq \emptyset$; otherwise, it is done already. Clearly, $P \subset A$, which implies $A \neq \emptyset$. Let B be the set of servers which sizes are at least $(k_s - 1)S$. Note that the three sets change during the execution of Algorithm 2(k_l, k_s).

Consider a server X in P. Recall the proof of Theorem 1. Before placing the last document d into X in Algorithm 1, X's load is less than $(k_l - 1)L$. Therefore, if we replicate d once in another server, d's load is at most $\frac{L}{2}$. Then, X's load will be less than $(k_l - 1)L + \frac{L}{2} = (k_l - \frac{1}{2})L$. Note that X's load will still be greater than $(k_l - \frac{1}{2})L - \frac{L}{2} = (k_l - 1)L$. So, X will still be in A. To guarantee that we can find a server for d's replica before the end of Algorithm 2, we need to prove the existence of a server which is not in $A \cup B$.

Before we do any replication, if $B = \emptyset$, then $|A \cup B| = |A| \leq \frac{M}{k_l - 1} < M$. It means that we can find a server for replication. If $B \neq \emptyset$ and shares a positive load, $|A| < \frac{M}{k_l - 1}$. Similarly, $|B| < \frac{M}{k_s - 1}$ because $A \neq \emptyset$, implying that $|A \cup B| \leq |A| + |B| < M$. Therefore, we can always find a server for the first replication. (Base case.) Suppose we can find servers to replicate the last documents of k servers in P, where $1 < k < |P|$. After placing these replicas, $|A| < \frac{1}{k_l - 1} M$; otherwise, the sum of the loads of all servers will be greater than the sum of the loads of all the documents and their replicas (if any). A contradiction. After these replications, these k servers will then be in A (although not in P). Let S_A be the sum of the sizes of servers in A. Assume the k replicas are also in A. $|B - A| < \frac{M}{k_s - 1}$; otherwise, the sum of the sizes of all servers will be greater than the sum of the sizes of all documents before replication (i.e., $(k_s - 1)S|B - A| + \frac{S_A}{2} > MS$). Assume that some replicas are not in A, and the sum of sizes of these replicas is $S_{\neq A}$. We have $S_{\neq A} \leq S_A$ because the original documents are all in A. $|B - A| < \frac{MS - S_A + S_{\neq A}}{(k_s - 1)S} \leq \frac{M}{k_s - 1}$. Hence, by the fundamental inequality, we have $|A \cup B| = |A| + |B - A| < M$. Then, we can find a server for replicating the last document of another server in P.

By induction, we can always find a server for replicating the last document of each server in P. □

Step 1 takes $O(N)$ time and Step 2 $O(M)$. There are at most $\frac{2}{2k_l - 1} M < M$ servers which loads are greater than $(k_l - \frac{1}{2})L$. So, Step 3 has $O(M)$ iterations, and each Step 3.x needs $O(1)$ time. Altogether, the time-complexity of Algorithm 2 (k_l, k_s) is $O(N + M) = O(N)$. A remark to this algorithm is that the load bound can be very close to $\frac{3L}{2}$, if we allow a large k_s.

3.3 Algorithm 3—Placement with Sorting

For simplicity, we have one more positive number $q \leq 1$ in the parameter of the following algorithm. The algorithm can be divided into two parts. The first part is from Step 1 to 8, which handles documents with loads greater than qL. This part sorts the documents and places them into servers in rounds. In the second part, we do not applying sorting. It is only for placing documents with load not exceeding qL in the greedy manner as in Algorithm 1. If we choose a small value for q, the first part will dominate. For large values of q, the second part takes the role. We will choose the best value of q later after Theorem 3.

Algorithm 3 (k_l, k_s, q):
1. Compute L and S;
2. Let D be a list of documents and $D := \emptyset$;
3. Take out the documents with loads greater than qL from input to D, and sort D in descending order;
4. Take out the remaining documents from input and append to D without sorting.
5. $d :=$ the first document of D;
6. $SERVER :=$ set of all M servers;
7. $round := 1$;
8. while ($D \neq \emptyset$) and (d's load is greater than qL)
8.1 Find a server $X \in SERVER$;
8.2 Place d into X;
8.3 $D := D - \{d\}$;
8.4 $SERVER := SERVER - \{X\}$;
8.5 If $SERVER = \emptyset$
8.5.1 $round := round + 1$;
8.5.2 If $round \leq \lfloor k_s \rfloor$

Recompute $SERVER$ such that it contains servers which loads are less than L;

8.5.3 If $round > \lfloor k_s \rfloor$
Recompute $SERVER$ such that it contains servers which loads are at most $[k_l - 1 - q']L$ with $q' = \lfloor k_s \rfloor q(k_l - 2)L$, and sizes less than $(k_s - 1)S$;

8.6 If $D \neq \emptyset$
$d :=$ the first document of D;

9. Recompute $SERVER$ such that it contains servers which loads are less than $(k_l - 1)L$ and sizes less than $(k_s - 1)S$;

10. while $D \neq \emptyset$

10.1 $d :=$ the first document of D;

10.2 Find a server $X \in SERVER$;

10.3 Place d into X;

10.4 $D := D - \{d\}$;

10.5 If (the load of X is at least $(k_l - 1)L$) or (size at least $(k_s - 1)S$)
$SERVER := SERVER - \{X\}$;

Definition 1. *A document is r-round if it is placed in some server in Step 8 when the value of the variable round is r.*

Step 8 can be applied because of the sorted property of the input. The existence of low-loaded servers in Step 8.5.2 is a benefit of the sorted feature. Step 10 is "borrowed" from Algorithm 1. We can see a large value of q may be beneficial in Step 8, but not in Step 9, and vice versa.

Lemma 1. *The server set $SERVER$ is nonempty whenever Step 8.1 is executed.*

Proof. Initially, $SERVER \neq \emptyset$ in Step 6. If $round \leq \lfloor k_s \rfloor$ in Step 8.5.2, then before placing all documents, there should always be some server(s) with load(s) less than L. Hence, $SERVER \neq \emptyset$. Suppose $round > \lfloor k_s \rfloor$ in Step 8.5.3. Let K be the set of servers which loads are greater than $[k_1 - 1 - q']L$. We argue that $|K| < \frac{M}{k_l - 1}$. We consider the case $K \neq \emptyset$; otherwise, it is trivial. By the property of sorted list, the loads of all servers which are not in K will be at least $\lfloor k_s \rfloor qL$. By direct counting, we have

$$LM \geq \sum_{i \in [1,N]} l_i > [k_l - 1 - q']L|K| + \lfloor k_s \rfloor q(M - |K|)L,$$

which implies $|K| < \frac{M}{k_l - 1}$. Moreover, the number of servers which sizes are at least $(k_s - 1)S$ is at most $\frac{M}{k_s - 1}$. By the fundamental inequality, in Step 8.5.3, we can find at least one server which load is at most $[k_l - 1 - q']L$ and size less than $(k_s - 1)S$. In other words, $SERVER \neq \emptyset$. □

Lemma 2. *Algorithm 3 can place all documents into servers and terminate.*

Proof. Consider the case that Step 8 terminates before placing all documents into servers. Since there are at most $\frac{M}{k_l - 1}$ servers having loads of at least $(k_l - 1)L$, and at most $\frac{M}{k_s - 1}$ servers having sizes of at least $(k_s - 1)S$, by the fundamental inequality, there always exists a server which load is less than $(k_l - 1)L$ and size less than $(k_s - 1)S$. Therefore, Step 10 can complete the placement.

For the case that Step 8 terminates after placing all documents in the servers, it is already done.

We now prove that Step 8 will terminate. By Lemma 1, we can always find a server X in Step 8.1. Since server X exists, Step 8.2 can place d in X and Step 8.3 can decrease the size of D by one for each iteration. Hence, there are at most $|D|$ iterations in Step 8. In other words, Step 8 will terminate. □

Lemma 3. *Immediately after placing a document into a server X in Step 10, X's load is less than $(k_l - 1 + q)L$ and size less than $k_s S$.*

Proof. In Step 10, we only place documents, which loads are at most qL, on the servers with loads less than $(k_l - 1)L$. Result follows. □

Lemma 4. $\forall 2 \leq r \leq \lfloor k_s \rfloor$, *if there exists a document of r-round in a server, then its load is at most $\frac{L}{r-1}$.*

Proof. Suppose that we are now to place an r-round document in server X. Since $2 \leq r \leq \lfloor k_s \rfloor$, before placing the r-round documents in X, the condition in Step 8.5.2 must be true. This condition was also true in all previous iterations of Step 8. Hence, we have already placed $r - 1$ documents in X, and the sum of their loads are at most L. By the pigeon hole principle, one of these documents will have a load of at most $\frac{L}{r-1}$. By the sorted property, the coming r-round document will have a load of at most $\frac{L}{r-1}$. □

Lemma 5. $\forall r \in [2, \lfloor k_s \rfloor]$, *immediately after placing an r-round document into a server X in Step 8, X's load is less than $(1 + \frac{1}{r-1})L$ and size less than $k_s S$.*

Proof. According to the algorithm, before placing this r-round document, X has a load of less than L. By Lemma 4, the load of an r-round document is at most $\frac{L}{r-1}$. Result follows. □

Lemma 6. $\forall r > \lfloor k_s \rfloor$, *immediately after placing an r-round document into a server X in Step 8, X's load is less than $[k_l - 1 - q' + \frac{1}{\lfloor k_s \rfloor}]L$ and size less than $k_s S$.*

Proof. According to the algorithm, X will have a load of $[k_l - 1 - q']L$ plus the load of this r-round document. By Lemma 4, this document has a load of at most $\frac{L}{\lfloor k_s \rfloor}$. Result follows. □

Theorem 3. *After execution of Algorithm 3, all documents can be placed, and each server's load is less than $\max(2L, (k_l - 1 + q)L, [k_l - 1 - q' + \frac{1}{\lfloor k_s \rfloor}]L)$ and size less than $k_s S$.*

Proof. By Lemma 2, we can place all documents by Algorithm 3. Consider a server X. If $Last(X)$ is placed in Step 10, by Lemma 3, X will have a load of $(k_l - 1 + q)L$. If $Last(X)$ is an r-round document placed in Step 8, where $r \in [2, \lfloor k_s \rfloor]$, by Lemma 5, X will have a load of less than $2L$. If $r > \lfloor k_s \rfloor$, by Lemma 6, X will have a load of $[k_l - 1 - q' + \frac{1}{\lfloor k_s \rfloor}]L$. □

For a given k_l, the last two terms can be minimized by setting q to $\frac{1}{\lfloor k_s \rfloor (1 + \lfloor k_s \rfloor (k_l - 2))}$. The upper bound of the servers' loads is simplified to $\max(2L, (k_l - 1 + q)L)$. Hence, for $k_l \geq 3 - q$, the bound of the loads of the servers is $(k_l - 1 + q)L$, which is less than $(k_l - \frac{1}{2})L$. For $k_l < 3 - q$, the bound of the loads of the servers is $2L$. In particular, if

$\frac{1}{k_l-1}+\frac{1}{k_s-1}=1$, $k_l \geq \frac{7+\sqrt{3}}{3}$ is equivalent to $k_l-1+q \geq 2$. Recall that in Algorithm 2, the load bound is $(k_l-\frac{1}{2})L$. In other words, while the size limit remains unchanged, for $k_l \geq 3-q$, the power of sorting is even greater than the replication technique in Algorithm 2. Since a full replication technique will share the load equally and give a load bound of L, one may hope for a better replication technique in future work. On the other hand, for $k_l < 3-q$, the bound is $2L$, which is optimal regardless of the power of sorting. It can be illustrated by the worst case: $\forall i \in [1,N]$, $l_i=\frac{MN}{N}$, $N=M+1$, and $M \geq 2$. The bound on the loads of the servers is approaching $2L$ when M is very large. All documents carry the same load and make sorting useless. Its means that the load bound $2L$ should be tackled by replication in general.

Here are some examples of the effect of different choices of k_l and k_s: When $k_l=\frac{5}{2}$ and $k_s=4$, we can bound the loads of the servers by $2L$. When $k_l=3$ and $k_s=3$, we can bound the loads by $\frac{25}{12}L$, while $q=\frac{1}{12}$. When $k_l=4$ and $k_s=\frac{5}{2}$, we can bound the loads by $\frac{37}{12}L$, while $q=\frac{1}{12}$.

We can find a duality between the bounds on the servers' loads and sizes in the proof of the above lemmas and theorem in this section. If we exchange the positions of load and size in Algorithm 3, we can easily get Theorem 4 in the following.

Theorem 4. *We can place all documents such that each server's load is less than k_lL and size less than* $\max(2S,(k_s-1+t)S)$, *where* $t=\frac{1}{\lfloor k_l \rfloor(1+\lfloor k_l \rfloor(k_s-2))}$.

Step 3 takes $O(N)$ time to transfer those documents with loads at least qL to the list D. If k_l and k_s are constants, q will also be a constant.[2] The time for sorting D will then be $O(M \log M)$. Obviously, Step 4 takes $O(N)$ time. In Step 8.4, we can put X into another set $SERVER2$ of servers if it satisfies the conditions on size and load as in Step 6. The set $SERVER2$ will then be $SERVER$ in the next round. Hence, we do not need Steps 8.5.2 and 8.5.3 to rebuild the server set. These steps are stated in the algorithm only for ease of discussion. The whole Step 8 can be done in $O(N)$ time. Step 10 takes $O(N)$ time. Therefore, the time-complexity for Algorithm 3 is $O(N+M \log M)$.

3.4 Algorithm 4—Placement with Sorting and Replication

In Sections 3.1 and 3.3, we show the trade off between the bounds on the loads and sizes of the servers when the bound of the loads is higher than $2L$ and that of the sizes higher than $2S$. However, $2L$ is an optimal bound that we can hope for without replication. We now combine the techniques of sorting and selection replication and replicate some documents to achieve a better load bound of $\max((\frac{3}{2}+2\delta)L,(k_l-1+q)L)$ with the same size limit, where

$$\delta=\begin{cases}\frac{k_s-2}{\lfloor k_s \rfloor(k_s+\lfloor k_s \rfloor-2)} & \text{if } k_s \leq 4\\ \frac{1}{4(k_s-1)} & \text{if } k_s > 4.\end{cases}$$

2. In practice, we will not assign a very large value to k_s. Increasing k_s from 16 to 32 will only result in a reduction of at most $\frac{4}{105}$ for k_l. However, in theory, if we do not assume k_l and k_s to be constant, we need $O(N \log N)$ time for Step 3, and the whole algorithm.

In particular, if $\frac{1}{k_l-1}+\frac{1}{k_s-1}=1$ and $k_s \leq 4$, then $\delta=q$. The load bound here is strictly less than $(k_l-\frac{1}{2})L$ which is given by Algorithm 2. If $k_l \geq 3-q$, it is already done by Algorithm 3. Therefore, we consider the case $k_l < 3-q$. Let p be $\max(\frac{3}{2}+2\delta, k_l-1+q)$.

Algorithm 4 (k_l,k_s,q):
1. Perform Algorithm 3 (k_l,k_s,q);
2. If $k_l < 3-q$
2.1 For every server X which load is at least pL
2.1.1 $d := Last(X)$;
2.1.2 Find a server $Y \neq X$ with minimum load and size less than $(k_s-1)S$;
2.1.3 Replicate d and put the replica into Y;

Theorem 5. *After execution of Algorithm 4, all documents can be placed, and each server's load is less than pL and size less than k_sS.*

Proof. By Lemma 2, Step 1 has placed all documents into servers. By Lemma 3, we are done for the case $k_l \geq 3-q$. Consider the other case $k_l < 3-q$. If a server X has a load greater than pL (i.e., greater than $(k_l-1+q)L$), $Last(X)$ must be placed in Step 8 in Algorithm 3. By Lemmas 5 and 6, if X has load ranging from pL to $2L$, exclusively, $Last(X)$ must be of 2-round. By the property of a sorted list, all other servers must have 1-round documents with a load not less than $(p-1)L$.

Let K be the set of servers having loads at least pL immediately after the execution of Step 1. If we replicate the (2-round) document in each server in K, the load of this server will drop to less than $\frac{3}{2}L$. This document and its replica will share half of the load, which is at most $\frac{L}{2}$. Altogether, we have $|K|$ new replicas. Thus, the problem is now to find $|K|$ other servers whose loads are less than $(p-\frac{1}{2})L$, and sizes less than $(k_s-1)S$, so that we can place the replicas into these servers, respectively, without exceeding the load bound pL.

Let P be the set of servers which loads are less than pL and at least $(p-\frac{1}{2})L$ immediately after the execution of Step 1. The loads of servers which are not in $K \cup P$ are at least $(p-1)L$, but less than $(p-\frac{1}{2})L$. By direct counting on the load of the whole system, we have

$$\begin{aligned}LM &\geq \sum_{i\in[1,N]} l_i\\ &\geq pL|K|+(p-\frac{1}{2})L|P|+(p-1)L(M-|K|-|P|),\end{aligned}$$

which implies $(2-p)M \geq |K|+\frac{|P|}{2}$. The number of servers which sizes are at least $(k_s-1)S$ is at most $\frac{M}{k_s-1}$. Therefore, the number of servers which loads are less than $(p-\frac{1}{2})L$ and sizes less than $(k_s-1)S$ is at least

$$\begin{aligned}&(M-|K|-|P|)-\frac{M}{k_s-1}\\ &\geq (1-\frac{1}{k_s-1}-(4-2p))M+|K| \geq |K|,\end{aligned}$$

while $2p-3-\frac{1}{k_s-1} \geq 0$ can be proven by checking two cases, $k_s \geq 4$ and $4 > k_s > 3$, separately. The validity of either case is deduced by substituting $k_l < 3-q$ to the fundamental inequality. □

This result improves the load bounds of Algorithms Two and Three. In particular, for $k_l = \frac{5}{2}$, $k_s = 4$, the load bound is $\frac{5}{3}L$; if $k_l = 2.25$, $k_s = 6$, the load bound will be $\frac{8}{5}L$.

Step 1 takes $O(N + M \log M)$ time. There are at most $O(M)$ iterations in Step 2. Step 2.1.2 will take $\log M$ steps to find a server which has the minimum load. It can be done by maintaining a heap of servers according to their loads. Hence, like Algorithm 3, the time-complexity of this algorithm is also $O(N + M \log M)$. It should be noted that without the constraint on sizes of the servers, we can achieve $\frac{3}{2}L$ as a bound for the loads.

3.5 Algorithm 5—Dynamic Simple Placement

The algorithms discussed in Sections 3.1, 3.2, 3.3, and 3.4 are offline and as such do not cater for dynamic insertion. The values of L and S in those algorithms are calculated before actual insertion. In this section, we consider dynamic placement, in which we do not know the total number of documents to be placed. As in Algorithm 1, we will not apply sorting and replication in this algorithm.

We redefine N to be the number of inserted documents, and $\forall i \in [1, N]$, l_i and s_i are the load and size of the ith inserted document, respectively. The values for L and S refer to the inserted documents only. We need to guarantee that after each insertion, the load and size of each server are bounded by some constants times L and S, respectively.

Algorithm 5 (k_l, k_s):
1. $S := 0$; L:=0;
2. Upon the arrival of a document d with load l and size s
2.1 Update L and S;
2.2 Find a server X with lowest possible load such that size $< (k_s - 1)S$;
2.3 Place d into X;

Using arguments similar to those in the proof of Theorem 1, the same bounds for load $k_l L$ and size $k_s S$ can still be obtained here. Consider the time complexity. Assuming addition and division can be done in constant time, Step 2.1 takes constant time, too, as updating the maximum and average of a set of values requires two additions and one division. As for Step 2.2, similar steps for finding a suitable server requires constant time in previous algorithms. This is because, in those algorithms, servers are tested against static load and size limits, L, $[k_l - 1 - q\lfloor k_s \rfloor (k_l - 2)]\ L$, $(k_l - 1)L$, and $(k_s - 1)S$, respectively. Once a server's load or size exceed their limits, they will always exceed the limits and the server can never accommodate any new documents again. However, this is not the case in this section as long as the values of L and S are dynamic and keep growing whenever any new documents arrive. A server, with load or size momentarily exceeding their limits, may be able to accommodate new documents later as the values of L and S rise to higher values. It means that we cannot exclude any servers permanently, even if they currently cannot be used for storing any new documents.

We define a tree, called B^0-tree of order K, to maintain all servers according to their loads and sizes, where $4 \leq K < M$. The purpose is to bound the time required to find a suitable server by $O(\log M)$. $O(\log M)$-time should also be enough to maintain the tree properties. We assign each server a unique identity. The B^0-tree stores only the servers' identities, not their content. We define $lo(x)$ to be the load of the server with identity x, and $si(x)$ is defined in a similar way for size. Each leaf node stores the identity of a server. Hence, we always have M leaf nodes.

Modifying from the B^+-tree, which was proposed by Knuth [16], a B^0-tree of order K has the following properties: The root node has 0, or 2 to K children. All nodes except the root and the leaf nodes have $\lceil \frac{K}{2} \rceil$ to K children. All leaf nodes are at the same level. Each node has a key. Recall that a leaf node stores the unique identity of a server. Its key is then assigned to be this identity. Each nonleaf node keeps a list of key-pointer pairs of its children. Formally, (x, y) is a (key, pointer) pair in a node A if x is the key of a child node B of A and y is pointing to B. The key x of an internal node A is chosen from the key of one of its children B such that for any child C of A, if $C \neq B$ and y is the key of C, then $si(x) \leq si(y)$. We call this property *the property of minimum size*.

The list of pairs is sorted in ascending order from left to right according to the loads that the key values correspond to. All the leaves form a sorted linear structure from left to right. Formally, the sorted list of key-pointer pairs in a node is $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$, where $\lceil \frac{K}{2} \rceil \leq k \leq K$, if for any identities u and v of nodes accessible from y_i and y_j, respectively, where $1 \leq i < j \leq k$, we have $lo(u) \leq lo(v)$. We call this property *the property of increasing load*. Fig. 1 shows an example. The document placement algorithm maintains implicit pointers for assessing the actual servers. In the B^0-tree, leaves are sorted according to the corresponding loads. Inside each noninternal node, a key is chosen according to the corresponding size.

To search for a suitable server, we start from the root. We scan the keys in the root list from left (smallest load) to right. Let x be the first key that can satisfy the size constraint. All descendant nodes of each key before x violate the size limit due to the property of minimum size. On the other hand, all descendant nodes of each key after x have loads at least as high as x due to the property of increasing load. If there is a suitable server that can be found under a key after x, then a suitable server must exist under x, too. Hence, we do not need to scan the keys after x. After choosing x, we stop scanning the root and go down to the child of x. In the child of x, we perform steps similar to those performed in the root. Eventually, we will find a suitable server at the leaf level. For example: We need to find a server of minimum load and a size of at most 31 (Fig. 1). Then, key D is chosen in the root, as server D gives the smallest load while $si(D)$ is bounded by the required size limit 31. Next, in the leftmost node in the second level, key G is chosen, as server G gives the smallest load while $si(G)$ is still bounded by 31.

After placing a document on a server, its load and size will both increase. Maintenance of the B^0-tree is like what we do for a B^+-tree, except that we may need to do both insertion and deletion at the same time. The reason for this is that after insertion the property of increasing load may be violated. If this is the case, we need to delete the corresponding leaf node and insert it to a proper position to preserve the property. (This position is found with the help of an auxiliary B^+-tree.) Like the B^+-tree, insertion and deletion can cause splitting and merging of nodes, respectively. Both of splitting and merging may induce a change the keys of the ancestors of the splitting/merging nodes because of the property of minimum size. An example of document placement is shown in Fig. 2.

In this example, we place a document in server G. After document placement, $lo(G)$ and $si(G)$ become 100 and 50,

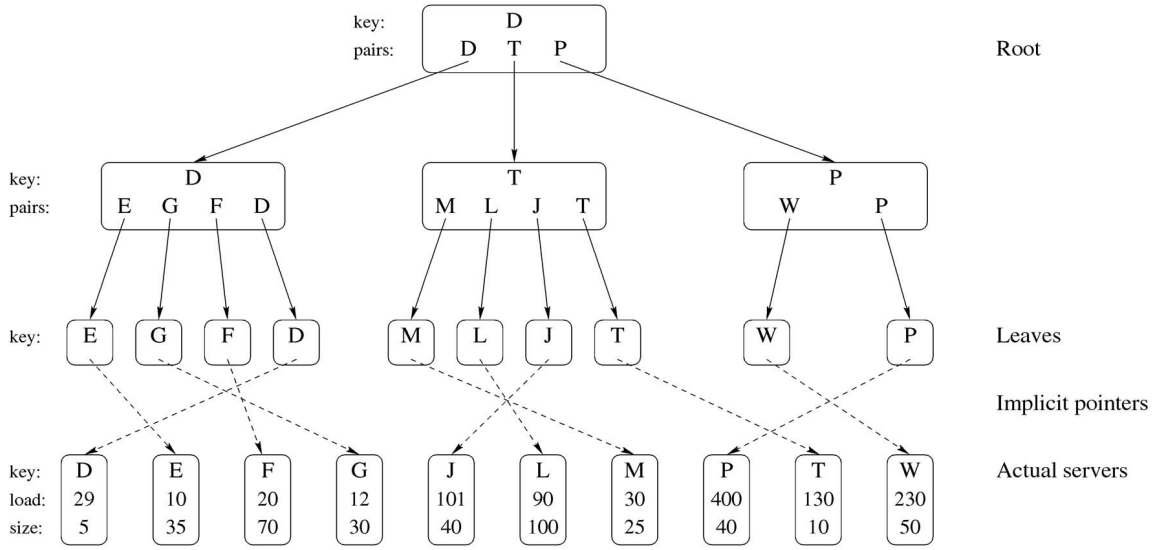

Fig. 1. An example of B^0-tree of order 4.

respectively, as shown in Fig. 2. The leaf node with key G needs to be moved to a new position, which is between L and J. Such a move induces restructuring of the B^0-tree.

Obviously, the depth of the B^0-tree is $O(\log_K M)$ and, in each node, it takes K steps to find a suitable identity in each level. Hence, to find a suitable server requires $O(K \log_K M)$ time. In the worst case, maintenance must traverse from the bottom node to the root twice. Both steps require $O(K \log_K M)$ time. (The auxiliary B^+-tree requires the same time-complexity, too.) By choosing $K = O(1)$, the time required to place one document is then $O(\log M)$. Therefore, to place N documents, the time-complexity is $O(N \log M)$. Compared with Algorithm 1, the factor $\log M$ is the price of the dynamic insertion.

4 Conclusion

In this paper, we make use of the fundamental inequality (1) to represent the relationship between the parameters k_l and k_s for the upper bounds of loads and sizes of the servers. Table 1 summarizes our results. The symbols q and t in the table represent $\frac{1}{\lfloor k_s \rfloor (1+\lfloor k_s \rfloor (k_l-2))}$ and $\frac{1}{\lfloor k_l \rfloor (1+\lfloor k_l \rfloor (k_s-2))}$. The parameter δ is $\frac{k_s-2}{\lfloor k_s \rfloor (k_s+\lfloor k_s \rfloor -2)}$ for $k_s \leq 4$, and $\frac{1}{4(k_s-1)}$ for $k_s > 4$.

Although we have only balanced the loads and sizes of the servers, we can modify the fundamental inequality in order to cope with more independent parameters such as the number of documents in each server, if necessary. If the parameters are dependent, however, the fundamental inequality cannot be applied directly.

In practice, in order to achieve the best bounds, we need to equalize both sides of the fundamental inequality. Then, the value of k_l decreases with the value of k_s. Balancing the loads of servers is a way to minimize the user waiting time. Where the user waiting time is critical, we need a smaller k_l.

Another practical concern is that a document's load may vary periodically, but not its size. In this case, we need a smaller k_s because size balancing is more rewarding than

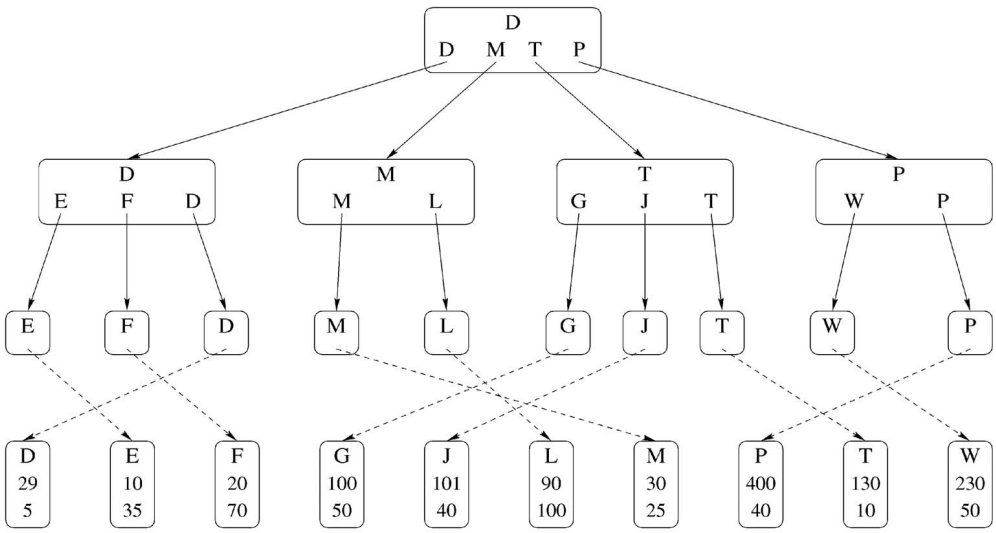

Fig. 2. An example of document placement.

TABLE 1
A Summary

Algorithms / Time-Complexities	Load Bounds ($\times L$)	Size Bounds ($\times S$)
One / $O(N)$	k_l	k_s
Two / $O(N)$	$(k_l - \frac{1}{2})$	k_s
Three / $O(N + M \log M)$	$\max(2, k_l - 1 + q)$	k_s
A Dual of Three	k_l	$\max(2, k_s - 1 + t)$
Four / $O(N + M \log M)$	$\max(\frac{3}{2} + 2\delta, k_l - 1 + q)$	k_s
Five / $O(N \log M)$	k_l	k_s

load balancing, as long as the load is still affordable. Another example is to store historical archive documents. We are more concerned with the size because the load of each document is normally low.

The practicability of the results in this paper is based on the simplicity of the algorithms. The basic operation of the algorithms is greedy approach which makes the algorithms easy to be handled. Moreover, the upper bounds in this paper act as a guarantee of the practical performance.

ACKNOWLEDGMENTS

The author thanks the anonymous referees for their very useful comments. In Algorithm 4 (Section 3.4), the value $\delta = \frac{1}{4(k_s-1)}$, for $k_s > 4$, was proposed by one of them. The preliminary version of this paper is in the Proceeding of the Seventh International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN 2004), pp. 61-66.

REFERENCES

[1] Akamai Technologies, Inc., http://www.akamai.com, 2005.
[2] D. Andresen, T. Yang, V. Holmedahl, and O.H. Ibarra, "SWEB: Towards a Scalable World Wide Web Server on Multicomputers," *Proc. 10th Int'l Parallel Processing Symp.,* pp. 850-856, Apr. 1996.
[3] M.F. Arlitt and T. Jin, "A Workload Characterization Study of the 1998 World Cup Web Site," *IEEE Network,* vol. 14, no. 3, pp 30-37, May/June 2000.
[4] T. Brisco, "DNS Support for Load Balancing," *RFC 1794, The Internet Eng. Task Force,* Apr. 1995.
[5] R.B. Bunt, D.L. Eager, G.M. Oster, and C.L. Williamson, "Achieving Load Balance and Effective Caching in Clustered Web Servers," *Proc. Fourth Int'l Web Caching Workshop,* Mar. 1999.
[6] V. Cardellini, E. Casalicchio, M. Colajanni, and P.S. Yu, "The State of the Art in Locally Distributed Web-Server Systems," *ACM Computing Surveys,* vol 34, no. 2, pp. 263-311, June 2002.
[7] V. Cardellini, M. Colajanni, and P.S. Yu, "DNS Dispatching Algorithms with State Estimators for Scalable Web-Server Clusters," *World Wide Web,* vol. 2, no. 3, pp. 101-113, 1999.
[8] V. Cardellini, M. Colajanni, and P.S. Yu, "Dynamic Load Balancing on Web-Server Systems," *IEEE Internet Computing,* vol. 3, no. 3, pp 28-39, May/June 1999.
[9] S. Ceri, G. Pelagatti, and G. Martella, "Optimal File Allocation in a Computer Network: A Solution Based on the Knapsack Problem," *Computer Networks,* vol. 6, pp. 345-357, 1982.
[10] L.C. Chen and H.A. Choi, "Approximation Algorithms for Data Distribution with Load Balancing of Web Servers," *Proc. IEEE Int'l Conf. Cluster Computing,* pp. 274-281, Oct. 2001.
[11] M. Colajanni, P.S. Yu, and V. Cardellini, "Dynamic Load Balancing in Geographically Distributed Heterogeneous Web Servers," *Proc. Int'l Conf. Distributed Computing Systems,* pp. 295-302, May 1998.
[12] M. Colajanni, P.S. Yu, and D.M. Dias, "Analysis of Task Assignment Policies in Scalable Distributed Web Server Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 6, pp. 585-600, June 1998.
[13] L.W. Dowdy and D.V. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys,* vol. 14, no. 2, pp. 287-313, 1982.
[14] M.L. Fisher and D.S. Hochbaum, "Database Location in Computer Networks," *J. ACM,* vol. 27, pp. 718-735, 1980.
[15] E.D. Katz, M. Butler, and R. McGrath, "A Scalable HTTP Server: The NCSA Prototype," *Computer Networks and ISDN Systems,* vol. 27, no. 2, pp. 155-164, 1994.
[16] D.E. Knuth, *The Art of Computer Programming,* vol. 3: Sorting and Searching, Section 6.2.4, Addison-Wesley, 1973.
[17] B. Narendran, S. Rangarajan, and S. Yajnik, "Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Access," *Proc. 16th Symp. Reliable Distributed Systems,* pp. 97-106, Oct. 1997.
[18] V.S. Pai, M. Aron, and G. Banga, "Locality-Aware Request Distribution in Cluster-Based Network Servers," *ACM SIGOPS Operating Systems Rev.,* vol 32, no. 5, pp. 205-216, Dec. 1998.
[19] R. Simha, B. Narahari, H.A. Choi, and L.C. Chen, "File Allocation for a Parallel Webserver," *Proc. Third Int'l Conf. High Performance Computing,* pp. 16-21, Dec. 1996.
[20] L. Zhuo, C.L. Wang, and F.C.M. Lau, "Load Balancing in Distributed Web Server Systems with Partial Document Replication," *Proc. 2002 Int'l Conf. Parallel Processing,* pp. 305-312, Aug. 2002.

Savio S.H. Tse received the BSc degree in physics in 1988 and the PhD degree in computer science in 1997 from the University of Hong Kong. After graduation, he was a visiting assistant professor at The University of Hong Kong for two years and a lecturer at The Hong Kong Polytechnic University for three years. Currently, he is a guest lecturer at The University of Hong Kong. His research interests are in the areas of parallel and distributed computing.