ONLINE BICRITERIA LOAD BALANCING FOR DISTRIBUTED FILE SERVERS

Savio Tse
Computer Engineering Department,
Bilkent University,
06800 Ankara, Turkey
Email: sshtse@cs.bilkent.edu.tr

***Abstract*—We study the online bicriteria load balancing problem in a system of M distributed homogeneous file servers located in a cluster. The load and storage space are assumed to be independent. We propose two online approximate algorithms for balancing the load and required storage space of each server during document placement.**

Our first algorithm combines the first result in [10] and the upper bound result in [1]. With applying document reallocation, we further obtain improvement and give a smoother tradeoff curve of the upper bounds of load and storage space. This result improves the best existing solutions. The second algorithm is for theoretical purpose. Its existence proves that the bounds for the load and the required storage space of each server, respectively, are strictly better when document reallocation is allowed. It enhances the research in applying document reallocation. The time complexities of both algorithms are $O(\log M)$; and the cost of document reallocation should be taken into account.

Keywords: Load balancing, Scheduling; Document placement, Re-allocation.

I. INTRODUCTION

The problem we address is to balance two independent parameters. It can be considered as a variant of the classical NP-complete *File Allocation Problem* (FAP). Based on the classical Knapsack Problem, which is also an NP-complete problem, Ceri et al. solved the optimal FAP in 1982 [2]. In [3], a survey given by Dowdy and Foster contains many results before 1982. In [9], we proposed five algorithms, including an $O(\log M)$-time online algorithm which bounds the load and storage space of each server by $k_l L$ and $k_s S$, respectively, where L and S are the optimal bounds for load and storage space, respectively, and $k_l > 2$, $k_s > 2$, and $\frac{1}{k_l-1}+\frac{1}{k_s-1} \leq 1$. In [1], Bilò et al. gave a $(\frac{2M-k}{M-k+1}, \frac{M+k-1}{k})$-competitive algorithm, where k can be any integer from 1 to M. It bounds the load and storage space by $\frac{2M-k}{M-k+1}L$ and $\frac{M+k-1}{k}S$, respectively. Note that there are M points for the choices of tradeoff between load and storage space. This result is originally for bicriteria scheduling problem and can be directly applied to load balancing. Asymptotically ($M \to \infty$), the bounds are the same as those of the online algorithm in [9], and a slight improvement for general values of M. In [10], we gave three algorithms. The first one is for placing documents in heterogeneous server systems. Formally, the jth server has bounds $p_l^j L$ and $p_s^j S$ for load and storage space, respectively, where $p_l^j, p_s^j > 2$, $p_l^j + p_s^j \geq 6$, and $j \in [1, M]$. It is asymptotically the best algorithm when it is applied to homogeneous servers. When the load and storage space are bounded by around three times of their optimal values, and M is small, the algorithm in [1] is better. We study them in Section III.

The load balancing problem is similar to the classical scheduling problem in many aspects. The latest result given by Fleischer and Wahl in [4], which is a $(1+\sqrt{\frac{1+\ln 2}{2}})$-competitive algorithm, can be applied to load balancing. For bicriteria scheduling, Rasala et al. gave many results in [8]. The first parameter is one of maximum flow time, makespan and maximum lateness, while the second is chosen from average flow time, average completion time, average lateness and number of on-time jobs. Since the two parameters are not independent, these results and techniques cannot be used to our problem.

In this paper, we design online algorithms for balancing (or scheduling) two independent parameters in homogeneous servers by allowing object re-allocation which has not been used for the existing results. We assume the extra cost is the sum of sizes of objects needed to migrate. This assumption is practical in our scenario of systems of distributed file servers, but may be too harsh for some scenario like balancing the load and the number of jobs assigned for each CPU in the shared memory model. Resource reallocation is a typical technique for load balancing. It can be applied to various kind of areas such as online processor scheduling [5], distributed memory management [6], etc.. Reallocation will inevitably impose extra communication cost in the network. Therefore, we need to keep it to a reasonable amount.

Our first result is a combination of the first algorithm in [10] and and the one in [1]. As expected, the algorithm having more benefit will be applied. Recalling that the algorithm in [1] allows M discrete points for the choices of tradeoff between load and storage space. By further reallocation, our contribution is to connect the discrete points concerned and give a new and smoother curve of tradeoff.

A lower bound result in [1] states that no (c_t, c_s)-competitive algorithm exists for the bicriteria scheduling problem, where $c_t < 2$ and $c_s < M$. However, our second result is to prove the existence of such values, for $M > 2$, under document reallocation. It clearly shows that document

reallocation is strictly beneficial in bicriteria load balancing.

The time complexity for all algorithms given in this paper is $O(\log M)$ plus the reallocation cost. The analyses do not include the physically placements of documents into the servers.

The paper is organized as follows: Section II gives the model and background data structures. The first algorithm is given in Section III, and the second one is in Section IV. Section V concludes our results and states some possible future research work.

II. Definitions and Models

Each document has two fundamental attributes, namely load and size. There are M homogeneous servers and N documents. The value of N changes upon each placement and deletion. The ith document has positive load l_i and size s_i, $\forall i \in [1, N]$. The load and storage space of a server is the summation of loads and sizes of all documents stored, respectively. For all $j \in [1, M]$, the load of the jth server is denoted as $\mathcal{L}_j$ and the storage space as $\mathcal{S}_j$. We do not assume any fixed limit on their values; however, there is still a need to balance them among the servers.

Let $\overline{L}$ and $\overline{S}$ be the average load and storage space of all servers in the system. Therefore, $\overline{L} = \frac{\sum_{i\in[1,N]} l_i}{M}$, and $\overline{S} = \frac{\sum_{i\in[1,N]} s_i}{M}$. Let L be $\max(\max_{i\in[1,N]} l_i, \overline{L})$ and S be $\max(\max_{i\in[1,N]} s_i, \overline{S})$. Clearly, L and S are the optimal bounds on the load and storage space of each server, respectively.

We apply a tree structure like B^+-tree [7] which is widely employed for storing the information of the servers throughout this paper. We call it *B^0-tree*. A B^0-tree stores a set $\{(x, y)|x, y \in R^+\}$. We assume the elements stored in a B^0-tree are unique[1]. Like B^+-tree, data (keys) are stored in leaves and all leaves are located at the bottom level. Except the root, each internal node has $\frac{K}{2}$ to K children. The root has 1 to K children. Like B^+-tree, the data in the bottom level are sorted according to y-values, and unlike B^+-tree, a parent node stores a copy of one of its children which has smallest x-value. If there are two children having the smallest x-value, choose the one with smaller y-value. Hence, the root contains the copy of the data with minimum x-value. An example is in Figure 1. Recall that a parent's x-value is no more than those of its children. We call this property *the property of minimum size*. Horizontally, at each level, the y-values are sorted from left to right. We call this property *the property of increasing load*. The normal node-splitting and merging operations are similar to B^+-tree. To keep the time for maintenance in $O(\log t)$, where t is the number of data stored in the tree, there is an auxiliary B^+-tree for storing the y-values only.

Let $\mathcal{A}^*$ be the algorithm for performing searching and updating on a B^0-tree. For any input (X, Y), where $X, Y \in R^+$, $\mathcal{A}^*$ can search an element (x, y) in a B^0-tree and perform updating within $O(\log t)$ time, where $x \le X$ and $y \le Y$. If there are two elements with smallest y-value, choose the one with smaller x-value. In the case that no suitable (x, y) in T, $\mathcal{A}^*$ will output false. The algorithm $\mathcal{A}^*$ is as follows: On input (X, Y), search from the root. If x-value is greater than X, return false. If $x \le X$, search for a children which x-value is at most X and y-value is minimum, and go to this child. Repeat this step until the bottom level. If the y-value of the last found node is greater than Y, return false; otherwise return the values of this node. For updating, we need an auxiliary B^+-tree for searching the bottom position. Then, proceed to update its parent until there is no need to update or the root is reached.

For conciseness, all B^0-trees used in this paper will be automatically updated and maintained unless specified.

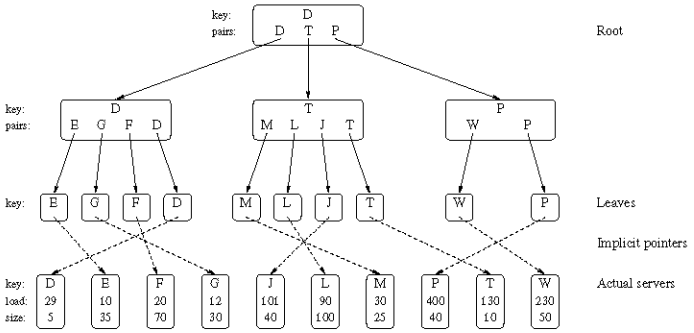

Fig. 1. An example of B^0-tree storing $\{(s_i, l_i)|i \in [1, 10]\}$.

III. The First Result

Consider the best existing online algorithm for bicriteria scheduling in [1], which is a (parametric) $(\frac{2M-k}{M-k+1}, \frac{M+k-1}{k})$-competitive algorithm, where $k \in \{1, 2, \ldots, M\}$. It is called Algorithm $A(k)$. It is designed for homogeneous server systems. We rewrite the load and storage space bounds in a different way as follows: The load and storage space of each server are bounded by $t_l L$ and $t_s S$, respectively, where $t_l = \frac{2M-k}{M-k+1}$ and $t_s = \frac{M+k-1}{k}$. In particular, when $k = 1$, $(t_l, t_s) = (2 - \frac{1}{M}, M)$, and when $k = 2$, $(t_l, t_s) = (2, \frac{M+1}{2})$. As there are M values for k, there are M pairs of (t_l, t_s) which can be used by Algorithm $A(k)$. It is easy to check that each of them satisfies $\frac{1}{t_l-1} + \frac{1}{t_s-1} = 1 + \frac{2}{M-1}$. We call the equation Curve B. We call $t_l + t_s = 6$ Line C. Line C represents the tradeoff between load and storage space, which is from the first algorithm in [10]. This algorithm is called H in this section. In applying Algorithm H, we set $t_l = p_l^j$ and $t_s = p_s^j$, $\forall j \in [1, M]$.

Consider Curve B and Line C. Their interaction points are $(3 - \Delta, 3 + \Delta)$ and $(3 + \Delta, 3 - \Delta)$, where $\Delta = \sqrt{\frac{8}{M+1}}$. It implies that for all (t_l, t_s) from $(2, \frac{M+1}{2})$ to $(3-\Delta, 3+\Delta)$, and from $(3 + \Delta, 3 - \Delta)$ to $(\frac{M+1}{2}, 2)$, along Curve B, Algorithm H outperforms $A(k)$. [2] We combine the advantages from each

[1] Precisely, we can organize the information in the format of $(B_1, B_2, \ldots, B_{M'})$, where $B_i = (x, y)$ for some $x, y \in R^+$, $\forall i \in [1, M']$

[2] For any two distinct points (x, y) and (x', y'), (x, y) outperforms (x', y') if and only if $x \le x'$ and $y \le y'$.

algorithm and form a new one. Figure 2 shows the new set of upper bound pairs.

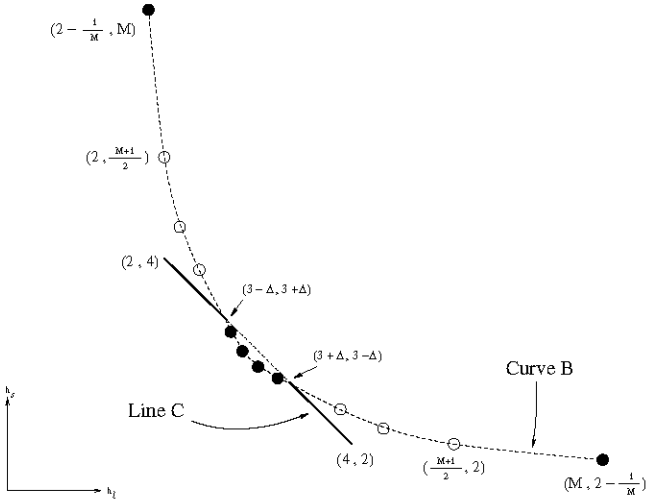

Fig. 2. The combined set of upper bound pairs.

In the figure, the resulting upper bound pairs are shown by the solid dots on Curve B, and two solid portions of Line C. The white dots on Curve B are not used as we can find better solution from Line C. The dotted portion of Line C is also unused, as the solid dots are better although discrete. Hence, there are five portions for (t_l, t_s) as follows:

1) $t_l = 2 - \frac{1}{M}$ and $t_s = M$.
2) $t_l + t_s = 6$, for $2 \le t_l \le 3 - \Delta$.
3) $\frac{1}{t_l - 1} + \frac{1}{t_s - 1} = 1 + \frac{2}{M-1}$ for $3 - \Delta < t_l < 3 + \Delta$ and there exists a $k \in \{1, 2, \ldots, M\}$ such that $t_l = \frac{2M-k}{M-k+1}$ and $t_s = \frac{M+k-1}{k}$.
4) $t_l + t_s = 6$, for $3 + \Delta \le t_l \le 4$.
5) $t_l = M$ and $t_s = 2 - \frac{1}{M}$.

Portions (2) and (4) are the only two continuous sets, and they are from Algorithm H. Portions (1) and (5) are the extreme cases from Algorithm $A(k)$. As little improvement of an additive term $\frac{1}{M}$ in one parameter does not justify the large cost of a factor $\frac{M}{4}$ on another one, probably these two pairs of values will not be used in practice[3]. Portion (3) contains many discrete points which can be used by Algorithm $A(k)$. Whether the other points within this range of Curve B can be used by any algorithms remain undetermined. The discrete nature may cause some inconvenience for the system designers to find the best load-storage space tradeoff. In this section, with the help of document reallocation, we bridge up the discrete points and form a continuous set of realizable upper bound pairs along Curve B of this portion, for all $M \ge 23$.

Recall that $3 - \Delta < t_l < 3 + \Delta$, each discrete point then corresponds to an integer k between $\frac{M+1}{2} - \sqrt{\frac{M+1}{2}}$ and $\frac{M+1}{2} + \sqrt{\frac{M+1}{2}}$. Consider another point (t_l, t_s) of Curve B within this portion. There exists an integer k laying between $\lfloor \frac{M+1}{2} - \sqrt{\frac{M+1}{2}} \rfloor$ to $\lfloor \frac{M+1}{2} + \sqrt{\frac{M+1}{2}} \rfloor$ such that

$$\begin{array}{l} \frac{2M-k}{M-k+1} < t_l < \frac{2M-k-1}{M-k} \\ \frac{M+k}{k+1} < t_s < \frac{M+k-1}{k} \end{array} \quad \text{and} \tag{1}$$

as (t_l, t_s) is surrounded by these two nearest points, $(\frac{2M-k}{M-k+1}, \frac{M+k-1}{k})$ and $(\frac{2M-k-1}{M-k}, \frac{M+k}{k+1})$, on B.

Upon the arrival of a document of load l and size s, if we can find a server which load is at most $\frac{M}{M-1}(t_l - 1)\overline{L}$ and storage space at most $\frac{M}{M-1}(t_s - 1)\overline{S}$, then the new document can be placed into this server and the resulting load is at most $t_l L$ and storage space at most $t_s S$, where $\overline{L}$ and $\overline{S}$ store the pre-placement values, and L and S store the post-placement ones. It is because the new load is at most $\frac{M}{M-1}(t_l - 1)\overline{L} + l = \frac{M}{M-1}(t_l - 1)(\overline{L}' - \frac{L}{M}) + l = \frac{M}{M-1}(t_l - 1)\overline{L}' + (1 - \frac{t_l - 1}{M-1})l \le \frac{M}{M-1}(t_l - 1)L + (1 - \frac{t_l - 1}{M-1})L = t_l L$, where $\overline{L}'$ is the new average load of the server. Similar argument to the storage space.

Let P be the set of servers which loads are more than $\frac{M}{M-1}(t_l - 1)\overline{L}$, and Q be the set of servers which storage space is more than $\frac{M}{M-1}(t_s - 1)\overline{S}$. Then, we have $|P| < \frac{M\overline{L}}{\frac{M}{M-1}(t_l - 1)\overline{L}} = \frac{M-1}{t_l - 1}$, and similarly, $|Q| < \frac{M-1}{t_s - 1}$. We try to find a server not in $P \cup Q$. If $P \cap Q \ne \emptyset$, then $|P \cup Q| = |P| + |Q - P| < \frac{M-1}{t_l - 1} + (\frac{M-1}{t_s - 1} - 1) = M + 1 - 1 = M$. That is, there is at least one server not in $P \cup Q$. If one of $\frac{M-1}{t_l - 1}$ and $\frac{M-1}{t_s - 1}$ is an integer, say $\frac{M-1}{t_s - 1}$, then $|Q| \le \frac{M-1}{t_s - 1} - 1$ and it implies the existence of one server outside $P \cup Q$, too.

Suppose that there is no server outside $P \cup Q$. In other words, $|P \cup Q| = M$. We then have $P \cap Q = \emptyset$ and both $\frac{M-1}{t_l - 1}$ and $\frac{M-1}{t_s - 1}$ are not integers. As $\frac{M-1}{t_l - 1} + \frac{M-1}{t_s - 1} = M + 1$, we have $\lfloor \frac{M-1}{t_l - 1} \rfloor + \lfloor \frac{M-1}{t_s - 1} \rfloor = M$. Since there is no available server for the new document, we apply document reallocation to vacate a server and Theorem 1 below shows this possibility. In practice, we simply take away the minimal number of documents to avoid the excessive reallocation cost.

Theorem 1: There exists an algorithm such that for all $M \ge 23$, it finds a server in P and a server in Q in $O(\log M)$ time such that the sum of their loads and their storage spaces are at most $t_l \overline{L}$ and $t_s \overline{S}$, respectively.

Proof: We first claim that

$$|P| = \lfloor \frac{M-1}{t_l - 1} \rfloor \text{ and } |Q| = \lfloor \frac{M-1}{t_s - 1} \rfloor. \tag{2}$$

Assume for contradiction that $|Q| < \lfloor \frac{M-1}{t_s - 1} \rfloor$. As $P \cap Q = \emptyset$, we have $M = |P \cup Q| = |P| + |Q|$. Recalling that $\lfloor \frac{M-1}{t_l - 1} \rfloor + \lfloor \frac{M-1}{t_s - 1} \rfloor = M$, we have $|P| > \lfloor \frac{M-1}{t_l - 1} \rfloor$, which implies $|P| \ge \lfloor \frac{M-1}{t_l - 1} \rfloor + 1 > \frac{M-1}{t_l - 1}$. This is a contradiction. If $|Q| > \lfloor \frac{M-1}{t_s - 1} \rfloor$, then $|Q| > \frac{M-1}{t_s - 1}$, which is also a contradiction. Therefore, $|Q| = \lfloor \frac{M-1}{t_s - 1} \rfloor$. We can use similar arguments for $|P|$.

Let $\delta_P \overline{S}$ be the total storage space of servers in P, and $\delta_Q \overline{L}$ be the total load of servers in Q. Taking δ_P and δ_Q into

[3]Theoretically, they remain two of the best pairs as there is no existing solution outperforms them.

account, we have

$$|P| < \frac{M\overline{L}-\delta_Q\overline{L}}{\frac{M}{M-1}(t_l-1)\overline{L}} = \frac{(M-\delta_Q)(M-1)}{M(t_l-1)}, \text{ and}$$
$$|Q| < \frac{M\overline{S}-\delta_P\overline{S}}{\frac{M}{M-1}(t_s-1)\overline{S}} < \frac{(M-\delta_P)(M-1)}{M(t_s-1)}. \quad (3)$$

We claim that $\frac{\delta_Q}{t_l-1} + \frac{\delta_P}{t_s-1} < \frac{1}{t_l-1} + \frac{1}{t_s-1}$. Assume the contrary, and by Equation (3), we have $M = |P \cup Q| = |P| + |Q| < (\frac{M-\delta_Q}{t_l-1} + \frac{M-\delta_P}{t_s-1})\frac{M-1}{M} \leq (\frac{M-1}{t_l-1} + \frac{M-1}{t_s-1})\frac{M-1}{M} = \frac{(M+1)(M-1)}{M} = M - \frac{1}{M}$. This is a contradiction, which implies that $\delta_Q < 1 + \frac{t_l-1}{t_s-1} < 1 + \frac{2+\Delta}{2-\Delta} < 2 + \frac{2\Delta}{2-\Delta} < 3$, for $M \geq 17$, and similarly, and $\delta_P < 3$.

The following algorithm is for searching the target servers for document reallocation. Let q be an integer less than $\min(|P|, |Q|)$, and its value will be determined later. Set integer $c = 0$. Loop $c = c + 1$ until the storage space of the cth smallest load server in P is no more than $\frac{\delta_P\overline{S}}{q}$. Output the cth smallest load server as X. Obviously, the loop terminates and Server X exists. Similarly, we find the c'th smallest storage space server Y in Q such that its load is no more than $\frac{\delta_Q\overline{L}}{q}$, and any server in Q, which has smaller storage space, has a load greater than $\frac{\delta_Q\overline{L}}{q}$. The load of X is less than $(\frac{M-\delta_Q-(c-1)\frac{M}{M-1}(t_l-1)}{|P|-(c-1)})\overline{L}$ and the storage space of Y is less than $(\frac{M-\delta_P-(c'-1)\frac{M}{M-1}(t_s-1)}{|Q|-(c'-1)})\overline{S}$. Both c and c' are no more than q, and the time complexity of this algorithm is $O(q \log M)$, which is $O(\log M)$ as q will be set as a constant.

Server X is vacated and its documents are reallocated to Y. The reallocation cost is at most $\frac{\delta_P\overline{S}}{q}$. The resulting load of Y is less than $(\frac{\delta_Q}{q} + \frac{M-\delta_Q-(c-1)\frac{M}{M-1}(t_l-1)}{|P|-(c-1)})\overline{L}$

$$\begin{aligned}
&\leq (\tfrac{\delta_Q}{q} + \tfrac{M-\delta_Q-(c-1)\frac{M}{M-1}(t_l-1)}{\lfloor\frac{M-1}{t_l-1}\rfloor-(c-1)})\overline{L} && \text{by (2)}\\
&\leq (\tfrac{\delta_Q}{q} + \tfrac{M-\delta_Q-(q-1)\frac{M}{M-1}(t_l-1)}{\lfloor\frac{M-1}{t_l-1}\rfloor-(q-1)})\overline{L}\\
&\quad \text{since } \lfloor\tfrac{M-1}{t_l-1}\rfloor < \tfrac{M-\delta_Q}{\frac{M}{M-1}(t_l-1)} \text{ and } c \leq q\\
&\leq (\tfrac{\delta_Q}{4} + \tfrac{M-\delta_Q-(3)\frac{M}{M-k+1}}{M-k-3})\overline{L}\\
&\quad \text{by (1), and setting } q = 4\\
&\leq (\tfrac{3}{4} + \tfrac{M-3-\frac{3M}{M-k+1}}{M-k-3})\overline{L}\\
&\quad \text{by } \delta_Q < 3 \text{ and } 4 < M-k-3\\
&\leq (1 + \tfrac{M-1}{M-k+1})\overline{L}\\
&\quad \text{by } k < \tfrac{M+1}{2} + \sqrt{\tfrac{M+1}{2}} \text{ and } M \geq 23\\
&< t_l\overline{L}, && \text{by (1)}
\end{aligned}$$

and by similar arguments, the resulting storage space is less than $t_s\overline{S}$. ■

We call the algorithm, which is guaranteed by Theorem 1, D. Since the theorem bridges the gaps between the discrete points, we now re-define portion (3) by removing the constraint of k and give an algorithm for this new portion as follows.

Algorithm $PORTION-THREE(t_l, t_s)$:
// $\frac{1}{t_l-1} + \frac{1}{t_s-1} = 1 + \frac{2}{M-1}$ and $3-\Delta < t_l < 3+\Delta$.
Upon the arrival of a document d
1. Perform Algorithm $\mathcal{A}^*$ on T_2 with input $(\frac{M}{M-1}(t_l-1)\overline{L}, \frac{M}{M-1}(t_s-1)\overline{S})$ and get output;
2. If output is $(\mathcal{L}_j, \mathcal{S}_j)$
2.1 Place d into the jth server;
3. If output is false
3.1 Perform D and output $X \subset P$, and $Y \subset Q$;
3.2 Move all documents in X to Y;
3.3 Place d into X;
4. Update $\overline{L}$ and $\overline{S}$;

For the discrete points used by Algorithm $A(k)$, Step 3 will not be executed and therefore no document reallocation is needed. Both P and Q are stored in B^0-trees.

A. Remarks

By studying the finite cases for M between 19 to 22, we can verify that the algorithm works for $M \geq 20$. By using the fact that δ_P and δ_Q are bounded by $2 + \frac{2\Delta}{2-\Delta} < 2.893$, we can improve the range of M a little bit. This kind of brute-force analyses is out of the interest of this paper, but it is useful to see the relationship between q and the range of M. In our algorithm, $M \geq 20$ when $q = 4$. If we choose $q = 5$, then $M \geq 25$, and $M \geq 29$ when $q = 6$, etc. Recall that the reallocation cost is bounded by $\frac{3\overline{S}}{q}$. In other words, for any M, we must use the greatest q. For the case that the value of M is a parameter during the system design, choosing the value of M is to decide the tradeoff between the reallocation cost and the other parameters such as maintenance cost incurred. (If M increases, the maintenance cost increases but the reallocation cost does not because q decreases.) As $M \to \infty$, if total storage space does not grow as fast as M, $\overline{S}$ will be very small and the reallocation cost is little.

IV. The Second Result

Theorem 2: There exists an online algorithm such that for any sequence of input, $\mathcal{L}_j \leq (2 - \frac{1}{M(M-1)})L$ and $\mathcal{S}_j \leq \frac{M+1}{2}S$, for all $j \in [1, M]$, $M > 2$.

Proof: Let X be the server of highest storage space before placement. If there are more than one choice for X, choose one arbitrarily. Placement of a new document in any server other than X does not violate the bound of storage space, as the new storage space of that server is at most

$$\frac{M\overline{S_0}}{2} + s = \frac{M}{2}(\overline{S'} - \frac{s}{M}) + s = \frac{M}{2}\overline{S'} + \frac{s}{2} \leq \frac{M+1}{2}S,$$

where $\overline{S_0}$ and $\overline{S'}$ stores the values of $\overline{S}$ before and after placement, respectively, and S stores the corresponding post-placement value.

It then suffices to prove that there exists a non-X server such that after placement into it, its load is bounded by $(2 - \frac{1}{M(M-1)})L$. Within the scope of this proof, we refer $\overline{L}$ and L to their post-placement values. Let l be the load of the document to be placed.

Assume that all servers, except the X, have load more than $(2-\frac{1}{M(M-1)})L-l$. Then, there is no available server before document reallocation. Let $\mathcal{L}_X$ be the load of X. Consider the total load after placement. Total load is at least $\mathcal{L}_X+l+(M-1)((2-\frac{1}{M(M-1)})L-l) = \mathcal{L}_X+ML-\frac{L}{M}+(M-2)(L-l)$. Hence, $\mathcal{L}_0 \leq \frac{L}{M}$ and $l > (1-\frac{1}{M(M-2)})L$; otherwise, total load is greater than ML, which is a contradiction. For the case $M \neq 2$, We then take out the documents in X, and place the new document into it. Now the load of each document taken out is at most $\frac{L}{M} < (1-\frac{1}{M(M-2)})L$. They can be placed into the servers without further document reallocation.

The data structures used are a heap for storing the information of servers according to their storage spaces, and a B^+-tree according to their loads. Each operation in these data structures needs at most $O(\log M)$ time. The highest storage space server is taken out from the B^+-tree, and will be back if another server has higher storage space. ■

V. Conclusion

We give two results for balancing the loads and storage spaces among servers during document placement.

Our first result is a combination of the first algorithm in [10] and Algorithm $A(k)$ in [1]. With document reallocation, it gives $\mathcal{L}_j \leq t_l L$ and $\mathcal{S}_j \leq t_l S$, for all $j \in [1, M]$, where $(t_l, t_s) \in \{(2-\frac{1}{M}, M), (M, 2-\frac{1}{M})\} \cup \{(t_l, t_s) | t_l + t_s \geq 6 \text{ or } \frac{1}{t_l-1}+\frac{1}{t_s-1} \geq 1+\frac{2}{M-1}\}$. Graphically, the contribution of this algorithm is a new curve of tradeoff as shown in Figure 2, with the middle discrete part smoothed. The reallocation cost is less than $\max(3\overline{S}, \hat{S})$, Let $\hat{S}$ be $\min(\mathcal{S}_i, S, 2s)$, $\mathcal{S}_i$ be the minimum storage space used among all servers, and s be the size of the incoming document. The reallocation cost is less than $\max(3\overline{S}, \hat{S})$, which is dominated by the algorithm in [10] because the additional document reallocation in the last algorithm is only $\frac{3\overline{S}}{q}$, where $q > 2$ is some integer increasing with M. Although the two points $(2-\frac{1}{M}, M), (M, 2-\frac{1}{M})$ are not very practical, they show two gaps at two ends which break the continuity in the final curve of t_l and t_s, as shown in Figure 2. It is of theoretical interest to bridge them. Further research can be done on it. Figure 2 also shows the large benefit obtained from document reallocation. This solution is based on a large value of M, which implies a small value of $\overline{S}$, and hence, a small reallocation cost. Further research is needed to reduce the reallocation cost for general values of M. Another research direction may be on different models of server heterogeneity.

From the first result, we have a wide range of choices for the system designers, and they can choose the most suitable according to their needs. For example, if the system is load sensitive, then the load bound would be better set to 2, while the storage space bound can be set to 4. How to choose a good tradeoff is the job of software engineers, and is out of the scope of the paper.

Our second result shows that when document reallocation is allowed, we can find an online algorithm for bounding the load and the required storage space of each server by $t_l L$ and $t_s S$, respectively, where $t_l < 2$ and $t_s < M$, or $t_l < M$ and $t_s < 2$. With the lower bound result [1] that no such values exist if document reallocation is not allowed, we conclude that document reallocation is absolutely advantageous. Much research is needed to explore its structures and properties which could enhance the practicality of document reallocation.

Acknowledgement

We thank the anonymous referees for their very useful comments.

References

[1] V. Bilò, M. Flammini, and L. Moscardelli, "Pareto Approximations for the Bicriteria Scheduling Problem", *Journal of Parallel and Distributed Computing*, vol. 66, No. 3, 393-402, 2006.

[2] 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, 345–357, 1982.

[3] L.W. Dowdy and D.V. Foster, "Comparative Models of the File Assignment Problem", *ACM Computing Surveys*, vol. 14, No. 2, 287–313, 1982.

[4] R. Fleischer and M. Wahl, "Online scheduling revisited", *Proceedings of the 8th Annual European Symposium on Algorithms (ESA)*, volume 1879 of Lecture Notes in Computer Science, 202-210. Springer Verlag, 2000.

[5] E. Haddad, "Runtime reallocation of divisible load under processor execution deadlines", *Proceedings of the Third Workshop on Parallel and Distributed Real-Time Systems*, 30–31, April 1995.

[6] H. Harada, Y. Ishikawa, A. Hori, H. Tezuka, S. Sumimoto, and T. Takahashi, "Dynamic home node reallocation on software distributed shared memory", *Proceedings of the Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region*, Vol. 1, 158–163, May 2000.

[7] D.E. Knuth, "The Art of Computer Programming, Vol. 3: Sorting and Searching, Section 6.2.4", *Addison-Wesley*, 1973.

[8] A. Rasala, C. Stein, E.Torng, and P. Uthaisombut, "Existence Theorems, Lower Bounds and Algorithms for Scheduling to Meet Two Objectives", *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 723–731, ACM Press, 2002.

[9] S.S.H. Tse, "Approximation Algorithms for Document Placement in Distributed Web Servers", *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, No. 6, 489–496, June 2005.

[10] S.S.H. Tse, "Online Solutions for Scalable File Server Systems", *Proceedings of the First International Conference on Scalable Information Systems (INFOSCALE 2006)*, Hong Kong, May 29–June 1, 2006.