Online Balancing Two Independent Criteria

Savio S.H. Tse

Department of Computer Engineering,
Bilkent University,
Ankara 06800, Turkey
sshtse@cs.bilkent.edu.tr
http://www.cs.bilkent.edu.tr/∼sshtse

Abstract. We study the online bicriteria load balancing problem in this paper. We choose a system of distributed homogeneous file servers located in a cluster as the scenario and propose two online approximate algorithms for balancing their loads and required storage spaces.

We first revisit the best existing solution for document placement, and rewrite it in our first algorithm by imposing some flexibilities. The second algorithm bounds the load and storage space of each server by less than three times of their trivial lower bounds, respectively; and more importantly, for each server, the value of at least one parameter is far from its worst case. The time complexities for both algorithm are $O(\log M)$.

Keywords: Approximate, Distributed, Online algorithm; Load balancing, Scheduling; Distributed file server; Document placement.

1 Introduction

Load balancing is a technique to achieve better coordination between entities such that the load burdened on each entity should not differ too much from that on others. In other words, load balancing is to prevent overwhelming any small subset of entities. The problem becomes NP-hard if we aim at evenly distributing the workload to all entities which provide the same services, or minimizing the difference between them. Therefore, approximate solutions are expected. The load on an entity can be its access rate, the number of execution of some important steps for each access, the number of bits transferred for each request, etc.. There are some different types of approximate solutions for load balancing. A common one is to bound the load of each entity by a limit [4,12,14]. Its variant is to set the limit according to the capacity of each individual entity [3]. In this paper, we choose the first type. In reality, there are often more than one parameter needed to be balanced. For example, execution time and memory utilization are two common parameters requiring simultaneous balancing. In this paper, we address the online bicriteria load balancing problem, and the two criteria are independent. We consider a system of distributed homogeneous file servers in a cluster, and the parameters to be balanced are the load and storage space. Hereafter, the single word "load" is referred to a parameter while "load

J. Cao et al. (Eds.): NPC 2008, LNCS 5245, pp. 244–254, 2008.

balancing" is referred to the classical problem. The load of a document stored in the file server system can be one of the quantities discussed above, and the storage space can be its physical size, or the memory space needed to process the document. The system designer can also take any other reasonable choices.

1.1 Related Works

With applying a limit to a set of homogeneous servers for bounding their loads, the single criterion load balancing problem is basically the NP-hard multiprocessor scheduling problem, which is reduced from the classical problem PARTITION [8]. Many heuristics have been proposed for solving it. The latest result was given by Fleischer and Wahl [7], which is an online $(1+\sqrt{\frac{1+\ln 2}{2}})$-competitive algorithm. (An online algorithm is c-competitive if the parameter needed to be minimized is bounded by c times its optimal values.) It is asymptotically the best known upper bound result. The latest lower bound result is by Rudin et al. [13], which shows that no c-competitive algorithm exists if $c < 1.88$.

For bicriteria load balancing, as there is one more constraint to tackle, higher upper bounds for both load and storage space are expected. In 2001, Chen et al. gave two offline algorithms, and one of them balances both the load and storage space [4]. It bounds the load by $4L$ using at most $4S$ storage space, where L and S (defined in Section 2) are commonly used as the trivial worst case lower bounds for load and storage space, respectively. In 2005, we proposed some algorithms [14], 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 $k_l > 2$, $k_s > 2$, and $\frac{1}{k_l-1} + \frac{1}{k_s-1} \leq 1$. In 2006, Bilò et al. gave a $(\frac{2M-k}{M-k+1}, \frac{M+k-1}{k})$-competitive algorithm [2], 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. It slightly improves the result for the online algorithm in [14], especially for small values of M. This is the best known result which can be generalized for balancing multi-parameters. Note that $k_l < 3 \Leftrightarrow k_s > 3$, and $\frac{2M-k}{M-k+1} < 3 - \frac{4}{M+1} \Leftrightarrow \frac{M+k-1}{k} > 3 - \frac{4}{M+1}$. Therefore, asymptotically ($M \to \infty$), there is no result which can bound the load by $h_l L$, and the storage space by $h_s S$, where h_l and h_s are any positive real numbers less than three.

1.2 Our Contribution

By modifying a technique in [14], we improve slightly on their last result in our first algorithm. This result is essentially the same as, but more flexible than, the upper bound result in [2]. The bounds of the load and storage space in our first result are $t_l L$ and $t_s S$, respectively, where $t_l, t_s > 1$ are real numbers satisfying both $\frac{1}{t_l-1} + \frac{1}{t_s-1} \leq 1 + \frac{2}{M-1}$ and $[\frac{M-1}{t_l-1}, \frac{M-1}{t_s-1} \notin I^+] \Longrightarrow [\lfloor \frac{M-1}{t_l-1} \rfloor + \lfloor \frac{M-1}{t_s-1} \rfloor < M]$. Comparing with the algorithm in [2], practically, the advantage of our algorithm is the flexibility in choosing suitable servers. An example in Section 3.2 shows the possibility of finding a server which can allow us to gain a lot in storage space

at the expense of little sacrifice on load. However, we improve the searching algorithm only for bicriteria load balancing, which is a special case for multicriteria load balancing tackled in [2].

We present our result in two equations, in which we can easily see the tradeoff between the upper bounds of load and storage space, and their symmetry and asymptotic behaviour (as $M \to \infty$). This representation has more theoretical benefit.

The last algorithm bounds the load and storage space of each server by $(3 - \frac{2}{M})L$ and $(3-\frac{2}{M})S$, respectively, with a feature that dictates if the load is higher than $(\frac{5}{2} - \frac{3}{2M})L$, then the storage space is less than $(\frac{5}{2} - \frac{3}{2M})S$ (and vice versa). In other words, at most one of load and storage space in each server can get close to their upper bounds. It is another style of load balancing, which does not exist in the literature [4,12,14], as far as we know.

2 Definitions and Models

Each document has two fundamental independent attributes, namely load and size. For the convenience of discussion, assume the load of a document to be the product of its access rate and its size plus the number of execution of some specific I/O steps. There are M servers and N documents. The value of N changes accordingly upon each placement and deletion. If server insertion is considered, The value of M will also increase by one on each server insertion. For every $i \in \{1, \ldots, N\}$, the ith document has positive load l_i and size s_i. For convenience, assume the indices of documents will automatically shift up upon each document deletion. The load and storage space of a server is the summation of loads and sizes of all documents stored, respectively. For all $j \in \{1, \ldots, M\}$, 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.

Let $\overline{L}$ and $\overline{S}$ be the average load and storage space of all servers in the system. Therefore, $\overline{L} = \frac{\sum_{i\in\{1,\ldots,N\}} l_i}{M}$, and $\overline{S} = \frac{\sum_{i\in\{1,\ldots,N\}} s_i}{M}$. As $\overline{S}$ is highly related to the upper bound of the cost of document recollocation, in order to keep its value reasonably small, M is assumed to be large enough although our algorithms also work for small M. Let L be $\max(\max_{i\in\{1,\ldots,N\}} l_i, \overline{L})$ and S be $\max(\max_{i\in\{1,\ldots,N\}} s_i, \overline{S})$. Note that $\overline{L}$, $\overline{S}$, L and S only depend on the existing documents stored and the number of servers. These algorithm-independent quantities are used in the descriptions of the upper bounds of $\mathcal{L}_j$ and $\mathcal{S}_j$, for all $j \in \{1, \ldots, M\}$, respectively, for all algorithms in this paper. Clearly, L and S are trivial lower bounds on the highest load and storage space of each server, respectively. For completeness, assume $\overline{L} = \overline{S} = 0$ and $L = S = 0$ when there is no document in the server system. We define the capacity index C_j for the jth server to be $\frac{\mathcal{L}_j}{L} + \frac{\mathcal{S}_j}{S}$, for each $j \in \{1, \ldots, M\}$. It is a metric that measures the combined effect of the loads and storage spaces of the servers, and the trivial lower bound of its worst case is obviously two. It is basically the sum of the normalized load and normalized storage space, and therefore, less affected by absolute values of two individual parameters. Obviously, $\sum_{j\in\{1,\ldots,M\}} C_j \leq 2M$.

The purpose of the capacity index is to enhance further balancing among servers. For example, if $\mathcal{L}_j \leq 3L$, $\mathcal{S}_j \leq 3S$, and $C_j < 4$, for all $j \in \{1, \ldots, M\}$, one can conclude that although the worst case of the load and storage space can be three times of L and S, respectively, only of them can be close to its worst case.

Let $t_l, t_s \in (1, M]$ be two real numbers satisfying both

$$\frac{1}{t_l - 1} + \frac{1}{t_s - 1} \leq 1 + \frac{2}{M-1}, \text{ and} \tag{1}$$

$$[\tfrac{M-1}{t_l-1}, \tfrac{M-1}{t_s-1} \notin I^+] \Longrightarrow [\lfloor \tfrac{M-1}{t_l-1} \rfloor + \lfloor \tfrac{M-1}{t_s-1} \rfloor < M]. \tag{2}$$

These two values are used throughout the paper to reflect the tradeoff between the bounds of loads and storage spaces for all servers. The relationship between t_l and t_s for all feasible pairs of values and the intuition of these two equations will be discussed later in Section 3.1. Fact 1 below will be used in some proofs in this paper.

Fact 1. *Suppose* $x_1, x_2 \in I^+$ *such that* $x_1 < \frac{M-1}{t_l-1}$ *and* $x_2 < \frac{M-1}{t_s-1}$. *Then,* $x_1 + x_2 < M$.

Proof. If both $\frac{M-1}{t_l-1}$ and $\frac{M-1}{t_s-1}$ are integers, then (let) $y = x_1 + x_2 \leq (\frac{M-1}{t_l-1} - 1) + (\frac{M-1}{t_s-1} - 1) < M$. If the former one (say) is an integer, then $y \leq (\frac{M-1}{t_l-1} - 1) + \lfloor \frac{M-1}{t_s-1} \rfloor = (\frac{M-1}{t_l-1} - 1) + (\lceil \frac{M-1}{t_s-1} \rceil - 1) \leq M + 1 - 2 < M$. If both of them are not integers, then $y \leq \lfloor \frac{M-1}{t_l-1} \rfloor + \lfloor \frac{M-1}{t_s-1} \rfloor < M$.

We apply a tree structure like B^+-tree [11] which is widely employed for storing the information of the servers in this paper. We call it B^0-*tree*, as [14]. A B^0-tree stores a set $\{(l, s) | l, s \in R^+\}$. In each order pair (l, s), l and s are referred to load and storage space of a server, respectively. We assume the elements stored in a B^0-tree are unique. (Precisely, the set can be organized as $(B_1, B_2, \ldots, B_{M'})$, where $B_j = (l, s)$ for some $l, s \in R^+$, $\forall j \in \{1, \ldots, M'\}$, $M' \leq M$.) Like B^+-tree, data (keys) are stored in leaves, and all leaves are located at the bottom level. Except for 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 s-values, and unlike B^+-tree, a parent node stores a copy of one of its children with smallest l-value. If there are two children having the smallest l-value, choose the one with smaller s-value. Hence, the root contains the copy of the data with minimum l-value. The normal operations are similar to those of 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 s-values only. For simplicity, we skip the discussion of those necessary but trivial steps for operations, like lookup, insertion and deletion on the data structure.

Let $\mathcal{SEEK}$ be the algorithm for performing searching and updating on a B^0-tree. This algorithm will be used in the following sections. For any input (X, Y), where $X, Y \in R^+$, $\mathcal{SEEK}$ can search an element (l, s) in a B^0-tree and perform updating within $O(\log t)$ time, where s is the smallest possible value such that $l \leq X$. If there are two l's with smallest s-value, choose the smaller one. In the case that $l > X$ for each (l, s) in the tree, $\mathcal{SEEK}$ will output false. The next step

is to check $s \leq Y$. If true, output (l, s); otherwise, output false. That means, if output is (l, s), then $l \leq X$ and $s \leq Y$. In other words, $\mathcal{SEEK}$ is used for searching a server with load and storage space inclusively bounded by certain values, respectively, and storage space is as less as possible.

By the similar construction, we can easily obtain an algorithm $\mathcal{SEEK}^*$ such that if output is (l, s), then $l < X$ and $s < Y$. In other words, $\mathcal{SEEK}^*$ is used for searching a server with load and storage space exclusively bounded by certain values, respectively, and storage space is as less as possible.

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

Let T_A be $\{(\mathcal{L}_j, \mathcal{S}_j) | j \in \{1, \ldots, M\}\}$ which is stored in a B^0-tree. That is, it stores the loads and storage spaces of all servers. The reallocation cost of a document is defined as its size. In particular, if all documents in the ith server are reallocated, the cost will be $\mathcal{S}_i$.

Lastly, our results are for synchronous networks; that is, before the completion of updating the data structures and reallocating the necessary documents for the previous operation, the next operation will not be performed.

3 The First Result

We consider document placement into a distributed file server. Our aim is to bound the loads and storage spaces of all servers by $t_l L$ and $t_s S$, respectively. With smaller values of t_l and t_s, the upper bounds are tightened and imply better balancing on load and storage space, respectively. The bounds are loosened slowly with M according to Equations (1) and (2). This matches with the fact that it is more difficult to coordinate more resources. However, such difficulty is not unlimited, as the bounds asymptotically tend to the result in [14]. We now apply tighter equations for t_l and t_s and analyse on the upper bounds.

Algorithm FIRST:
1. Upon the arrival of a document d with load l and size s
1.1 Perform $\mathcal{SEEK}$ on T_A with input $(\frac{M}{M-1}(t_l - 1)\overline{L}, \frac{M}{M-1}(t_s - 1)\overline{S})$ and get output $(\mathcal{L}_j, \mathcal{S}_j)$;
1.2 Place d into the jth server;
1.3 Update $\overline{L}$ and $\overline{S}$;

Theorem 1. *The new document can be placed into a server, and after placement, the load and storage space of the server are no more than $t_l L$ and $t_s S$, respectively.*

Proof. If the server system is initially empty, the algorithm can place the document and give the bounds L and S, respectively.

Assume there are some documents in the server system. Before placing the document d, there are less than $\frac{M-1}{t_l - 1}$ servers with load more than $\frac{M}{M-1}(t_l - 1)\overline{L}$,

otherwise, the total load will exceed $M\overline{L}$. Similarly, there are less than $\frac{M-1}{t_s-1}$ servers with storage space more than $\frac{M}{M-1}(t_s-1)\overline{S}$. By Fact 1, the number of servers exceeding the load bound or the storage space bound is less than M. Hence, there exists one server with load and storage space at most $\frac{M}{M-1}(t_l-1)\overline{L}$ and $\frac{M}{M-1}(t_s-1)\overline{S}$, respectively, and $\mathcal{SEEK}$ will output such a server as the jth server in Step 1.1.

Suppose that the average load is $\overline{L}'$ after Step 1.2. Then, $\overline{L}' = \overline{L} + \frac{l}{M}$. $\mathcal{L}_j$ is then at most $\frac{M}{M-1}(t_l-1)\overline{L}+l = \frac{M}{M-1}(t_l-1)(\overline{L}'-\frac{l}{M})+l = \frac{M}{M-1}(t_l-1)\overline{L}'+(1-\frac{t_l-1}{M-1})l$. The result for load follows as $\overline{L}'$ and l are no more than the final L. By using similar arguments, the result for storage space follows.

3.1 The Feasible Region for Values of t_l and t_s

We discuss the feasible region for values of t_l and t_s satisfying Equations (1) and (2). The purpose is to provide more information to the system designer to choose values for t_l and t_s for different situations.

For the case that $\frac{1}{t_l-1} + \frac{1}{t_s-1} \leq 1 + \frac{1}{M-1}$, Equations (1) and (2) are always true. The region for this case is labeled as A in Figure 1.

For the case that $\frac{1}{t_l-1} + \frac{1}{t_s-1} = 1 + \frac{2}{M-1}$, if $\frac{M-1}{t_l-1}$ and $\frac{M-1}{t_s-1}$ are non-integers, then $\lfloor\frac{M-1}{t_l-1}\rfloor + \lfloor\frac{M-1}{t_s-1}\rfloor = M$, which implies that Equation (2) is false. Then, we cannot use Fact 1 to guarantee the existence of a server for placement. In order to keep Equation (2) true, one of $\frac{M-1}{t_l-1}$ and $\frac{M-1}{t_s-1}$ must be an integer. As $\frac{M-1}{t_l-1} + \frac{M-1}{t_s-1} = M+1$, both $\frac{M-1}{t_l-1}$ and $\frac{M-1}{t_s-1}$ are integers between 1 and M, inclusively. In other words, there are M feasible pairs of t_l and t_s on the curve $\frac{1}{t_l-1} + \frac{1}{t_s-1} = 1 + \frac{2}{M-1}$, satisfying Equation (2). Let $k = \frac{M-1}{t_s-1}$. Then, $\frac{M-1}{t_l-1} = M-k+1$. Rewriting the result in Theorem 1 in terms of M and k, our load bound $t_lL = \frac{2M-k}{M-k+1}L$, and storage space bound $t_sS = \frac{M+k-1}{k}S$. This matches exactly with the $(\frac{2M-k}{M-k+1}, \frac{M+k-1}{k})$-competitive algorithm in [2]. In other words, if we equalize the inequality in Equation (1), the algorithm FIRST has identical upper bounds as in [2]. As k is ranged from 1 to M, there are M feasible points for (t_l, t_s) on the curve $\frac{1}{t_l-1} + \frac{1}{t_s-1} = 1 + \frac{2}{M-1}$.

Claims 3.1 and 3.1 below investigate the structure for t_l and t_s satisfying $1 + \frac{1}{M-1} < \frac{1}{t_l-1} + \frac{1}{t_s-1} < 1 + \frac{2}{M-1}$.

Claim. For all $t_l, t_s \in R^+$ satisfying $1 + \frac{1}{M-1} < \frac{1}{t_l-1} + \frac{1}{t_s-1} < 1 + \frac{2}{M-1}$, Equation (2) is true if and only if there exists a $k \in \{1, 2, \ldots, M\}$ such that $t_s \geq \frac{M+k-1}{k}$ and $t_l \geq \frac{2M-k}{M-k+1}$.

Proof. Suppose Equation (2) is true. If one of $\frac{M-1}{t_l-1}$ and $\frac{M-1}{t_s-1}$ is an integer, without loss of generality, assume that $\frac{M-1}{t_s-1}$ is an integer, and let $k = \frac{M-1}{t_s-1}$. Then $t_s = \frac{M-1}{k} + 1$. As $\frac{M-1}{t_l-1} + \frac{M-1}{t_s-1} < M+1$, we have $\frac{M-1}{t_l-1} < M+1-k$, and result follows. If both $\frac{M-1}{t_l-1}$ and $\frac{M-1}{t_s-1}$ are non-integers, then $\lfloor\frac{M-1}{t_l-1}\rfloor + \lfloor\frac{M-1}{t_s-1}\rfloor < M$. Let $k = \lfloor\frac{M-1}{t_s-1}\rfloor$. Then $\frac{M-1}{t_l-1} - 1 < \lfloor\frac{M-1}{t_l-1}\rfloor < M-k$, and result follows.

Suppose Equation (2) is false. Then, both $\frac{M-1}{t_l-1}$ and $\frac{M-1}{t_s-1}$ are non-integers and $\lfloor \frac{M-1}{t_l-1} \rfloor + \lfloor \frac{M-1}{t_s-1} \rfloor = M$. For all $k \in \{1, 2, \ldots, M\}$, we have $t_s \geq \frac{M+k-1}{k} \Leftrightarrow k \geq \frac{M-1}{t_s-1} > \lfloor \frac{M-1}{t_s-1} \rfloor \Leftrightarrow M-k < \lfloor \frac{M-1}{t_l-1} \rfloor \Leftrightarrow M-k+1 \leq \lfloor \frac{M-1}{t_l-1} \rfloor < \frac{M-1}{t_l-1} \Leftrightarrow t_l < \frac{2M-k}{M-k+1}$. Result follows.

Claim. For all $k \in \{1, 2, \ldots, M-1\}$, the point $(\frac{2M-k-1}{M-k}, \frac{M+k-1}{k})$ is on the curve $\frac{1}{t_l-1} + \frac{1}{t_s-1} = 1 + \frac{1}{M-1}$.

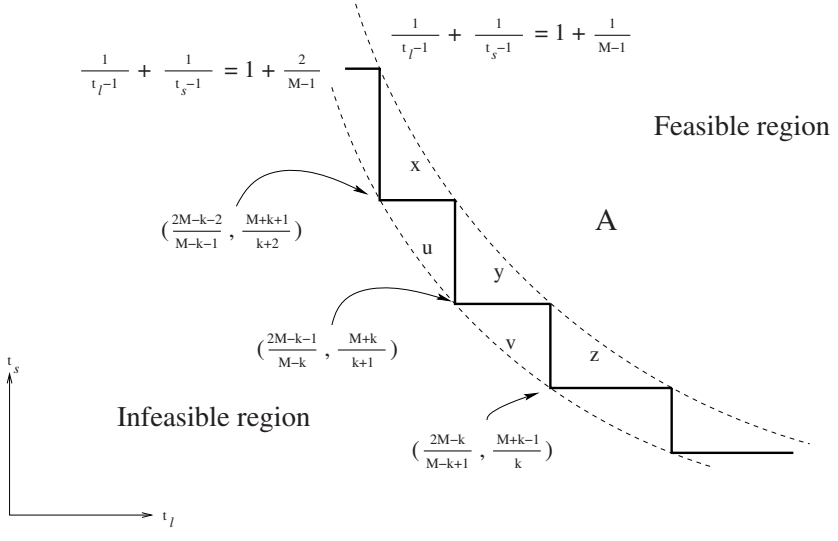

Fig. 1. Feasible Region for values of t_l and t_s

We skip the trivial proof for Claim 3.1. Recalling the M feasible points is on the curve $\frac{1}{t_l-1} + \frac{1}{t_s-1} = 1 + \frac{2}{M-1}$, and together with Claims 3.1 and 3.1, the whole feasible region is now clear and is shown in Figure 1. In the figure, the feasible and infeasible regions are separated by the solid zigzag (-horizontal-vertical-) line which is bounded tightly by two dotted curves $\frac{1}{t_l-1} + \frac{1}{t_s-1} = 1 + \frac{1}{M-1}$ and $\frac{1}{t_l-1} + \frac{1}{t_s-1} = 1 + \frac{2}{M-1}$, and the vertices of the zigzag line touch the curves alternatively. The feasible region is divided into two types of sub-regions. The largest sub-region is the open-ended one bounded below by the upper curve. We label it as A. The sub-regions of the second type, which are disjoint, spread over the gap between two curves. The ones labeled as x, y and z are examples. The values inside the sub-regions of this type satisfy Equation (2). In contrast, the points between two adjacent sub-regions turn Equation (2) false. Examples are u and v in the figure. It is easily seen that the M feasible points in the lower curve are the best in the whole feasible region. Precisely, for every other point in the feasible region, there is a better choice from these M feasible points. Since the two curves will narrow and become one as $M \to \infty$, the sub-regions of the

second type diminish with M, and the M best points will coincide with the upper curve.

From the feasible region, we have some suggestions to the system designer. First if M is unchanged, we can use one of the M best points on the lower curve. After the system designer chooses a point, he can proceed to check if both Equations (1) and (2) remain true. If yes, he can apply his values. Otherwise, use binary search to find a point out of the M best points, which is nearest to his original choice. The time needed is $O(\log M)$. Binary search can be used because of the convex nature of the feasible region.

If M can be increased by server insertion, the previous M points may become infeasible as the lower curve shifts upwards. Even when M decreases by server deletion, some points may fall into infeasible region when the two curves shifts down. One can easily see from the figure that there are no two consecutive points staying in the feasible region as M decreases by 1. In order not to put burden on the system maintenance, we suggest to use the points, satisfying $\frac{1}{t_l-1}+\frac{1}{t_s-1} \leq 1$, in the region A, if M can change. These points, used in [14], are independent of M, and is suitable for a system in which the number of servers is changing.

3.2 Remarks on Algorithm FIRST

Comparing Algorithm FIRST with the result in [2], our algorithm has two advantages. First, our upper bounds can spread through the continuous feasible region, not only the M best points. The second advantage comes from the difference of searching algorithms. The algorithm in [2] ignores the servers of the first $M-k$ highest load, $k \in \{1,\ldots,M\}$. In the case that the loads of some of these ignored servers are not very high but the storage spaces of them are very low, our algorithm is beneficial. Take for an example. For all $j \in \{1,\ldots,M-k-1\}$, $\mathcal{L}_j = \overline{L}+\delta$ and $\mathcal{S}_j = \frac{M\overline{S}-\delta}{M-1}$; $\mathcal{L}_{M-k} = \overline{L}$ and $\mathcal{S}_{M-k} = \delta$; and for all $j \in \{M-k+1,\ldots,M\}$, $\mathcal{L}_j = \overline{L} - \frac{M-k-1}{k}\delta$ and $\mathcal{S}_j = \frac{M\overline{S}-\delta}{M-1}$, where δ is extremely small. Then, one of the last k servers will be chosen by the algorithm in [2], but algorithm $\mathcal{SEEK}$ will choose the $(M-k)$th one. The former have little advantage on load but pays much higher price on storage space. Nevertheless, the searching algorithm in [2] can be easily generalized for balancing more than two criteria. Although $\mathcal{SEEK}$ is better, it is designed for two criteria only. Further research can be done on finding better searching algorithms for multi-criteria load balancing problem.

4 The Second Result

In this section, we study the capacity index which measures the integrated effect of load and storage space on each server. Our aim is to bound the load, the storage space and the capacity index of each server by $(3-\frac{2}{M})L$, $(3-\frac{2}{M})S$, and $5-\frac{3}{M}$, respectively, after each document placement.

Consider the algorithm FIRST. We choose $t_l = t_s = 3-\frac{4}{M+1}$ for odd M. For even M, we choose $t_l = 3-\frac{2}{M}$, and $t_s = 3-\frac{6}{M+2}$, or vice versa. Then, a

trivial upper bound $6-\frac{8}{M+1}$ on the capacity index can be obtained immediately. In this section, by using algorithm SECOND, the capacity index is improved to $5-\frac{3}{M}$, at the expense of a slightly higher upper bound(s) for load and/or storage space, respectively. In other words, if we sacrifice the asymptotically nothing in the upper bounds of load and storage space, respectively, then we gain much more in capacity index in return.

Directly from the definition, the capacity index $5-\frac{3}{M}$ implies that for each server, at most one of the two parameters, load and storage space, can be close to its upper bound of worst case. For example, if the load in a server gets very close to $(3-\frac{2}{M})L$, then its storage space keeps a distance of nearly S from the upper bound $(3-\frac{2}{M})S$. In other words, by using algorithm SECOND, the worst cases of load and storage space are shared by more servers. However, by using algorithm FIRST, the load and storage space can both simultaneously reach their upper bounds, respectively. Therefore, algorithm SECOND beats FIRST when t_l and t_s are chosen close to three.

The improvement in capacity index also gives hope that both parameters could be very close to 2.5 of their trivial lower bounds simultaneously. If succeed, it will then an important step towards the asymptotic latest known upper bound of 1.9201 [7] and the lower bound of 1.88 [13] for balancing a single parameter.

As there always exists a $j \in \{1,\ldots,M\}$ such that $\mathcal{L}_j \leq 2\overline{L}$, $\mathcal{S}_j \leq 2\overline{S}$, and $\overline{C}_j \leq 2$ (otherwise, $\sum_{j=1,\ldots,M} \overline{C}_j > 2M$), an $O(M)$-time algorithm can be applied to search this server in order to obtain a better upper bound on capacity index. For small M, the average storage space is large, and this trivial approach is a better choice. However, when M is large, an $O(\log M)$-time algorithm $CAPACITY$ will be given. Its idea is as follows: Upon the arrival of a new document d, if there is a server in which load and storage space are bounded by $\overline{L}$ and $2\overline{S}$, respectively, Step 1.1 of the algorithm will find it and Step 1.2.1 will place d into it. After placement, the load, the storage space, and the capacity index are kept under the mentioned bounded. The details are shown in Theorem 2. Suppose no such server exists in the system. We aim at a server in which load and storage space are bounded by $2\overline{L}$ and $\overline{S}$, respectively. It such a server exists, Step 1.3.1 will find it out and Step 1.3.2 will place d into it. The correctness proof is based on the observation that if Step 1.1 fails in searching a server, then Step 1.3.1 will succeed.

The algorithm SECOND is given below, and is followed by Theorem 2.

Algorithm SECOND:
1. Upon the arrival of a document d with load l and size s
1.1 Perform $\mathcal{SEEEK}^*$ on T_A with input $(\overline{L}, 2\overline{S})$ and get output;
1.2 If output is $(\mathcal{L}_j, \mathcal{S}_j)$;
1.2.1 Place d into the jth server;
1.3 If output is false
1.3.1 Perform $\mathcal{SEEEK}^*$ on T_A with input $(2\overline{L}, \overline{S})$ and get output $(\mathcal{L}_i, \mathcal{S}_i)$;
1.3.2 Place d into the ith server;
1.4 Update $\overline{L}$ and $\overline{S}$;

Theorem 2. *The new document can be placed into a server, and after placement, the load and storage space of the server are less than* $(3 - \frac{2}{M})L$ *and* $(3 - \frac{2}{M})S$*, respectively, and the capacity index less than* $5 - \frac{3}{M}$.

Proof. Assume for contradiction that for all $j \in [1, M]$, $\mathcal{L}_j \geq 2\overline{L}$, $\mathcal{S}_j \geq 2\overline{S}$, or $[\mathcal{L}_j > \overline{L}$ and $\mathcal{S}_j > \overline{S}]$. Suppose there are M_1 servers which loads are at least $2\overline{L}$, M_2 servers which storage spaces are at least $2\overline{S}$, and M_3 servers which loads are more than $\overline{L}$, and storage spaces more than $\overline{S}$. Obviously, $M_1 + M_2 + M_3 \geq M$. If $M_1 = 0$, total storage space will exceed $M\overline{S}$. Hence $M_1 \neq 0$. Similarly, $M_2 \neq 0$. Consider that $M_3 = 0$. Since all servers have positive loads, total load is greater than $2M_1\overline{L}$, which implies $M_1 < M_2$. On the other hand, since all servers have positive storage spaces, total storage space is greater than $2M_1\overline{S}$, which implies $M_2 < M_1$. Hence, $M_3 \neq 0$. Considering $\sum_{j=1}^{M}[\frac{\mathcal{L}_j}{\overline{L}} + \frac{\mathcal{S}_j}{\overline{S}}] > 2M_1 + 2M_2 + 2M_3 \geq 2M$, which is a contradiction. Therefore, there exists a $j \in [1, M]$, such that $\mathcal{L}_j < 2\overline{L}$, $\mathcal{S}_j < 2\overline{S}$, and $[\mathcal{L}_j \leq \overline{L}$ or $\mathcal{S}_j \leq \overline{S}]$. Rewriting it, we have either $[\mathcal{L}_j \leq \overline{L}$ and $\mathcal{S}_j < 2\overline{S}]$ or $[\mathcal{S}_j \leq \overline{S}$ and $\mathcal{L}_j < 2\overline{L}]$. We assume the former case while the argument for the latter one is similar.

After placing d into the server, the average load becomes $\overline{L}' = \overline{L} + \frac{l}{M}$, the average storage space becomes $\overline{S}' = \overline{S} + \frac{s}{M}$, and the values of L and S become L' and S'. Then, $\mathcal{L}_i \leq \overline{L}' - \frac{l}{M} + l = \overline{L}' + (1 - \frac{1}{M})l \leq (2 - \frac{1}{M})L'$. For storage space, $\mathcal{S}_i < 2(\overline{S}' - \frac{s}{M}) + s = 2\overline{S}' + (1 - \frac{2}{M})s \leq (3 - \frac{2}{M})S'$. Hence, $C_i < 5 - \frac{3}{M}$.

References

1. Amita, G.C.: Incremental data allocation and reallocation in distributed database systems. Data warehousing and web engineering, 137–160 (2002)
2. Bilŏ, V., Flammini, M., Moscardelli, L.: Pareto Approximations for the Bicriteria Scheduling Problem. Journal of Parallel and Distributed Computing 66(3), 393–402 (2006)
3. Brinkmann, A., Salzwedel, K., Scheideler, C.: Compact, Adaptive Placement Schemes for Non-Uniform Requirements. In: Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2002), Winnipeg, Manitoba, Canada (August 2002)
4. Chen, L.C., Choi, H.A.: Approximation Algorithms for Data Distribution with Load Balancing of Web Servers. In: Proc. of IEEE International Conference on Cluster Computing, Newport Beach, CA, USA, pp. 274–281 (October 2001)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. McGraw-Hill, New York (2001)
6. Fisher, M.L., Hochbaum, D.S.: Database Location in Computer Networks. Journal of ACM 27, 718–735 (1980)
7. Fleischer, R., Wahl, M.: Online scheduling revisited. Journal of Scheduling, Special Issue on Approximation Algorithms for Scheduling Algorithms (part 2) 3(6), 343–353 (2000)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
9. Haddad, E.: Runtime reallocation of divisible load under processor execution deadlines. In: Proceedings of the Third Workshop on Parallel and Distributed Real-Time Systems, April 1995, pp. 30–31 (1995)

10. Harada, H., Ishikawa, Y., Hori, A., Tezuka, H., Sumimoto, S., Takahashi, T.: Dynamic home node reallocation on software distributed shared memory. In: Proceedings of the Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, May 2000, vol. 1, pp. 158–163 (2000)
11. Knuth, D.E.: The Art of Computer Programming. Sorting and Searching, Section 6.2.4, vol. 3. Addison-Wesley, Reading (1973)
12. Narendran, B., Rangarajan, S., Yajnik, S.: Data Distribution Algorithms for Load Balanced Fault-Tolerant Web Access. In: Proc. of the 16th Symposium on Reliable Distributed Systems, Durham, NC, USA, pp. 97–106 (October 1997)
13. Rudin III, J.F.: Improved bounds for the online scheduling problem. PhD thesis, The University of Texas at Dallas (2001)
14. Tse, S.S.H.: Approximation Algorithms for Document Placement in Distributed Web Servers. IEEE Transactions on Parallel and Distributed Systems 16(6), 489–496 (2005)