Online Bicriteria Load Balancing Using Object Reallocation

Savio S.H. Tse

Abstract—We study the bicriteria load balancing problem on two independent parameters under the allowance of object reallocation. The scenario is a system of M distributed file servers located in a cluster, and we propose three online approximate algorithms for balancing their loads and required storage spaces during document placement. The first algorithm is for heterogeneous servers. Each server has its individual trade-off of load and storage space under the same rule of selection. The other two algorithms are for homogeneous servers. The second algorithm combines the idea of the first one and the best existing solution for homogeneous servers. Using document reallocation, we obtain a smooth trade-off curve of the upper bounds of load and storage space. The last one 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 of these three algorithms are $O(\log M)$ plus the cost of document reallocation.

Index Terms—Scheduling, distributed file server, document placement, heterogeneous computing, nonuniform requirement.

1 Introduction

Internet and network services nowadays are unquestionably essential. In order to achieve reliability, scalability, efficiency, and availability, we cannot rely on stand-alone servers to provide internet services. Such inadequacy is commonly filled by distributed solutions due to the higher computing power and fault tolerance with graceful degradation. However, having more resources often leads to more difficulties in coordinating entities. If coordination is not done well, there can be much waste in resources. For example, the computing power of most servers in a distributed Web server system is wasted, if only the documents inside few servers are popular. Thus, the need for efficient systems can be converted into the need for better coordinations between entities.

Load balancing is a common technique to achieve better coordination between entities. It aims at distributing the workload to entities as evenly as possible. The problem becomes NP-hard if an optimal distribution of the workload is required, and, therefore, approximate solutions are expected. The load on an entity can be its access rate, the number of executions of important steps for each access, the number of bits transferred for each request, etc.. There are different types of approximate solutions for load balancing. One common type is to bound the load of each entity by a limit [2], [5], [13]. Its variant is to set the limit according to the capacity of each individual entity [3]. These are often referred to as homogeneous and heterogeneous systems, respectively. In reality, there is often more than one parameter to be balanced. For example, execution time and memory utilization are two common parameters requiring simultaneous balancing. In this paper, we address the problem of online balancing two independent criteria with the allowance of a limited amount of object reallocations. Object reallocation is referred to moving a subset of objects to their new positions within the system. The scenario is a system of distributed file servers in a cluster, and the parameters to be balanced are the load and storage space. 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, its compressed physical size, or the effective memory pages it occupies. The system designer can also make other reasonable choices.

1.1 Related Works

The problem we address is a variant of the classical NP-hard *File Allocation Problem* (FAP) [6]. It is to allocate files to entities in a distributed environment for optimizing a certain performance metric. Based on the classical Knapsack Problem, which is also NP-hard, Ceri et al. solved the optimal FAP in 1982 [4]. As expected in solving an NP-hard problem, this optimal solution takes exponential time and is not feasible as an online solution. A survey given by Dowdy and Foster [6] contains many results before 1982.

Here are some results for bicriteria load balancing in homogeneous servers without applying document reallocation. Chen and Choi [5] gave two algorithms and one of them 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 [13], we proposed four offline and one online algorithms. The first offline one is to place the document into any server in which load and storage space, respectively, are below certain levels. The other three offline algorithms use document replication, sorting, and a combination of both, respectively. The online algorithm

• *The author is with the Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey.*
E-mail: sshtse@cs.bilkent.edu.tr and sshtse@gmail.com.

Manuscript received 8 Nov. 2007; revised 16 Apr. 2008; accepted 25 Apr. 2008; published online 15 May 2008.
Recommended for acceptance by D. Trystram.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2007-11-0414.
Digital Object Identifier no. 10.1109/TPDS.2008.79.

uses the greedy technique; it executes in $O(\log M)$ time and bounds the load and storage space of each server by $k_l L$ and $k_s S$, respectively, where k_l, $k_s > 2$, $\frac{1}{k_l-1} + \frac{1}{k_s-1} \leq 1$, and M is the number of servers. Bilò et al. [2] 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 trade-off between load and storage space. This result is originally for bicriteria scheduling problem and can be directly applied to load balancing. Asymptotically, the bounds are the same as those from the online algorithm in [13].

The load balancing problem is similar to the classical scheduling problem in many aspects. In particular, the minimization of the overall completion time for a set of independent tasks in the scheduling problem is basically the same as the minimization of the upper bound of loading in the load-balancing problem. The latest result given by Fleischer and Wahl [7], 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 [11]. The first parameter is maximum flowtime, makespan, or maximum lateness; whereas the second one is average flowtime, average completion time, average lateness, or number of on-time jobs. The two parameters are not independent; thus, these results and techniques cannot be used for our problem.

For a single parameter, a common model of heterogeneity is to allow for different capacities among servers. Brinkmann et al. proposed an algorithm for balancing the number of identical objects in a system of servers of various capacities [3]. They applied adaptive hashing for reducing the time complexity and assigning more balls to a server of higher capacity, proportionally, with high probability. In contrast, our algorithms are deterministic, and the heterogeneity among servers is reflected by their individual load and storage space bounds.

Reallocation is a typical technique for load balancing and is used in this paper. It can also be applied in areas such as distributed database systems [1], online processor scheduling [8], or distributed memory management [9]. As extra communication cost is inevitably imposed on the network, it should be kept to a reasonable amount.

1.2 Our Contribution

In this paper, we design online algorithms for balancing (or scheduling) two independent parameters by allowing object reallocation, which has not been used for the existing results in the literature. The time complexities for a single document placement in all algorithms given in this paper are $O(\log M)$ plus the reallocation cost. We assume that the reallocation cost is the sum of the sizes of objects needed to migrate. This assumption is practical in our scenario of clusters of distributed file servers as reallocation cost is almost directly proportional to the number of bits transferred. Moreover, upon each document placement in our first two algorithms, the documents to be reallocated are grouped together into a bucket; and at most two buckets are needed in the third algorithm. A bucket is an atomic unit for reallocation. Document reallocation will then be done at most twice, and it does not push our algorithm to offline.

Our first result is an algorithm for placing documents in heterogeneous server systems. After each placement, for all $j \in \{1, \ldots, M\}$, the load and storage space of the jth server are bounded by $p_l^j L$ and $p_s^j S$, respectively, where p_l^j, p_s^j are any real constants satisfying $[p_l^j \geq 2$ and $p_s^j \geq 3]$ or $[p_l^j \geq 3$ and $p_s^j \geq 2]$. The cost for document reallocation is bounded by three times of the average storage space $\overline{S}$.

For homogeneous servers, the second result is designed for at least 18 servers. It combines the ideas of our first result and the algorithm of Bilò et al. Recall that their algorithm allows M discrete points for the choices of trade-off between load and storage space, and these points are lying on a curve, as discussed later. Except for two extreme points and at most two points in the middle portion of the curve, almost all these discrete points are improved by our first algorithm. With a different strategy for document reallocation, the middle portion is smoothed into a continuous segment. The cost for document reallocation is bounded by $\frac{3\overline{S}}{q}$, where $q \geq 4$.

The last algorithm bounds the load and storage space of each server by $3L$ and $3S$, respectively, with a feature that dictates if the load is higher than $2L$, then the storage space is less than $2S$ (and vice versa). Namely, at most one of load and storage space in each server can be higher than twice its trivial lower bound. The cost for document reallocation is bounded by $1.5\overline{S}$. The constant factor 1.5 can be improved to $1 + \alpha$ at the expense of a factor of $\frac{1}{\alpha}$ in the time complexity, where $0 < \alpha < \frac{1}{2}$.

The technique of document reallocation cannot be used directly in geographically distributed server systems in which the communication overhead is no longer predictable. As uncertainty is inevitable in this kind of systems, further research could be done on reasonable models and the feasibility for applying document reallocation on them.

1.3 Remarks on Practicability

The first algorithm in this paper has $2M$ parameters, namely, p_l^j and p_s^j, $j \in \{1, \ldots, M\}$, which satisfy the corresponding equation specified later. In considering the system requirements and constraints, the system designer can determine their values, which are intuitively a trade-off between load and storage space for each individual server. For example, a smaller p_l^j means that load is more important or that some resources concerning load are very tight.

The practicability of our results is based on their simplicity and online property. By simplicity, we mean that their implementations are not difficult. By online, we mean that the time cost for each operation is reasonably bounded, and the reallocation costs are reasonable, as $\overline{S}$ decreases with M. In the case of documents of small loads or sizes, the performance must be much better than our upper bounds, and the solution to such systems must be simpler. However, unless all special cases are filtered out, proven upper bounds are still needed as a guarantee of the practical performance.

1.4 Organization of the Paper

Section 2 gives the system model and related definitions. The first algorithm is in Section 3, and the second one is in Section 4. Section 5 is for the third algorithm, and it is

followed by a refinement for tuning down the reallocation cost in Section 5.1. Section 6 concludes our results and states some possible directions for future research.

2 Definitions and Models

Each document has two fundamental attributes, namely, load and size. The load of a document can be measured by its access rate, and the size can be its physical size. There are M servers and N documents. The value of N increases by one upon each placement. The ith document has positive load l_i and size s_i, $\forall i \in \{1, \ldots, N\}$. The load and storage space of a server is the summation of the 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 $\max(\max_{i\in\{1,\ldots,N\}} s_i, \overline{S})$. Clearly, L and S are the trivial lower bounds on the highest load and storage space of each server, respectively. 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 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 the 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 one of them can be close to its worst case. For convenience, we also define C-value $\overline{C}_j$ to be $\frac{\mathcal{L}_j}{\overline{L}} + \frac{\mathcal{S}_j}{\overline{S}}$, for each $j \in \{1, \ldots, M\}$. Obviously, $\sum_{j\in\{1,\ldots,M\}} \overline{C}_j = 2M$ and $C_j \leq \overline{C}_j$, for each $j \in \{1, \ldots, M\}$.

We apply a tree structure like B^+-tree [10], which is widely employed in this paper for storing the information of the servers. We call it B^0-tree, as in [13]. A B^0-tree stores a set $\{(x, y)|x, y \in R^+\}$. The values of x and y in each pair can be taken, in any order, from the load, storage space, and their variants, of a server. We assume that the elements stored in a B^0-tree are unique. (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, $y \in R^+$, $\forall i \in \{1, \ldots, M'\}$.) As in 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 y-values, and unlike B^+-tree, a parent node stores a copy of one of its children with the smallest x-value. If there are two children with the smallest x-value, choose the one with smaller y-value. Hence, the root contains the copy of the datum with minimum x-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 y-values only. For simplicity, we skip the discussion of those trivial steps for operations like lookup, insertion, and deletion on the data structure.

Let $\mathcal{SE}$ be the algorithm for performing searching and updating on a B^0-tree. For any input $X \in R^+$, $\mathcal{SE}$ can search an element (x, y) in a B^0-tree and perform updating within $O(\log t)$ time, where y is the smallest possible value such that $x < X$. If there are two elements with the smallest y-value, choose the one with smaller x-value. In other words, for any other element (x', y') in T such that $x' < X$, we have either $y' > y$ or $[y' = y$ and $x \leq x']$. In the case that $x \geq X$ for each $(x, y) \in T$, $\mathcal{SE}$ will output false. Intuitively, if the servers' information is stored as (load, storage space), $\mathcal{SE}$ can be used to find the server of minimum storage space with load bounded by a certain value. Similar to (storage space, load).

Another algorithm $\mathcal{SE}^*$ is used to search for (x, y), for any input (X, Y), where $x \leq X$ and $y \leq Y$. Like $\mathcal{SE}$, both algorithms update x and y, if needed, and output false if searching is not successful. Each algorithm takes $O(\log t)$ time. These two basic algorithms are used to simplify the pseudocodes in this paper. For conciseness, all B^0-trees used in this paper are automatically updated and maintained, unless specified.

Our results are for synchronous networks; that is, before the completion of updating the data structures and reallocating the necessary documents for the previous input document, the next input is not read. Last, the reallocation cost of a document is defined as its size. In particular, reallocating all documents in the jth server, $j \in \{1, \ldots, M\}$, needs cost $\mathcal{S}_j$.

3 Document Placement for Heterogeneous Servers

For all $j \in \{1, \ldots, M\}$, let p_l^j, p_s^j be two numbers in R, satisfying

$$\left[p_l^j \geq 2 \text{ and } p_s^j \geq 3\right] \text{ or } \left[p_l^j \geq 3 \text{ and } p_s^j \geq 2\right]. \quad (1)$$

The problem is to bound the load and storage space of the jth server by $p_l^j L$ and $p_s^j S$, respectively, $\forall j \in \{1, \ldots, M\}$, upon each document placement. Define $T_1 = \{(\frac{\mathcal{L}_j}{p_l^j - 1}, \frac{\mathcal{S}_j}{p_s^j - 1})|j \in \{1, \ldots, M\}\}$, $T_2 = \{(\mathcal{L}_j, \mathcal{S}_j)|j \in \{1, \ldots, M\}\}$, and $T_3 = \{(\mathcal{S}_j, \mathcal{L}_j)|j \in \{1, \ldots, M\}\}$, which are stored in separate B^0-trees.

For each input document, algorithm HETER below first searches for a server with load and storage space bounded by $(p_l^j - 1)\overline{L}$ and $(p_s^j - 1)\overline{S}$, respectively. If found (in Step 2), place the new document into this server. Assume no such server exists. Find a server X, indexed j in Step 3.1, of minimum storage space S_X with load bounded by $\overline{L}$, and a server Y, indexed k in Step 3.2, of minimum load L_Y with storage space bounded by $\overline{S}$. We now have four cases. Case 1 is $[S_X < 2\overline{S}$ and $L_Y < 2\overline{L}]$. Swap their contents, and

both of them can accept the new document. Case 2 is [$S_X < 2\overline{S}$ and $L_Y \geq 2\overline{L}$]. Find a server Z, indexed i in Step 3.4.1, of minimum storage space, and transfer a minimal subset of documents to Y such that Z can accept the new document. Case 3 is [$S_X \geq 2\overline{S}$ and $L_Y < 2\overline{L}$], which is similar to Case 2. Case 4 is [$S_X \geq 2\overline{S}$ and $L_Y \geq 2\overline{L}$], which is impossible.

Algorithm $HETER(p_l^1, p_s^1, p_l^2, p_s^2, \ldots, p_l^M, p_s^M)$:
Upon the arrival of a document d with load l and size s
1. Perform $\mathcal{SE}^*$ on T_1 with input $(\overline{L}, \overline{S})$ and get output;
2. If output is $(\frac{\mathcal{L}_j}{p_l^j - 1}, \frac{\mathcal{S}_j}{p_s^j - 1})$
2.1 Place d into the jth server;
3. If output is false
3.1 Perform $\mathcal{SE}$ on T_2 with input $\overline{L}$ and get output $(\mathcal{L}_j, \mathcal{S}_j)$;
3.2 Perform $\mathcal{SE}$ on T_3 with input $\overline{S}$ and get output $(\mathcal{S}_k, \mathcal{L}_k)$;
3.3 If $\mathcal{S}_j < 2\overline{S}$ and $\mathcal{L}_k < 2\overline{L}$ // case (1)
3.3.1 Swap the content of the jth and kth servers;
3.3.2 Place d into the jth server;
3.4 If $\mathcal{S}_j < 2\overline{S}$ and $\mathcal{L}_k \geq 2\overline{L}$ // case (2)
3.4.1 Perform $\mathcal{SE}$ on T_2 with input ∞ and get output $(\mathcal{L}_i, \mathcal{S}_i)$;
3.4.2 Take out a minimal subset R of documents from the ith server such that the total load in R is at least $\min(\mathcal{L}_i, L, l)$;
3.4.3 Place R into the jth server; Place d into the ith server;
3.5 If $\mathcal{S}_j \geq 2\overline{S}$ and $\mathcal{L}_k < 2\overline{L}$ // case (3)
3.5.1 Perform $\mathcal{SE}$ on T_3 with input ∞ and get output $(\mathcal{S}_i, \mathcal{L}_i)$;
3.5.2 Take out a minimal subset R of documents from the ith server such that the total size in R is at least $\min(\mathcal{S}_i, S, s)$;
3.5.3 Place R into the kth server; Place d into the ith server;
4. Update $\overline{L}$ and $\overline{S}$;

When a new document comes, if there exists a $j \in \{1, \ldots, M\}$ such that $\mathcal{L}_j \leq (p_l^j - 1)\overline{L}$ and $\mathcal{S}_j \leq (p_s^j - 1)\overline{S}$, Step 1 outputs it, and the new document is placed into it in Step 2.1. The load and storage space of the server are then no more than $(p_l^j - \frac{p_l^j - 1}{M})L$ and $(p_s^j - \frac{p_s^j - 1}{M})S$, respectively, where L and S are referred to their postplacement values. To avoid any ambiguities, we assume that L and S are updated with $\overline{L}$ and $\overline{S}$ in the algorithms proposed in this paper, unless specified otherwise.

Consider the case in which no such server exists. That means, for all $j \in \{1, \ldots, M\}$, we have

$$\mathcal{L}_j > \left(p_l^j - 1\right)\overline{L} \quad \text{or} \quad \mathcal{S}_j > \left(p_s^j - 1\right)\overline{S}. \tag{2}$$

Therefore, there exist a server with load less than $\overline{L}$ and a server with storage space less than $\overline{S}$, and Steps 3.1 and 3.2 will not output false.

Lemma 1. *Just after the execution of Step 3.2, we have $\mathcal{S}_j < 2\overline{S}$ or $\mathcal{L}_k < 2\overline{L}$, where $j \neq k$.*

Proof. Recall that $\mathcal{L}_j < \overline{L}$ and $\mathcal{S}_k < \overline{S}$. If $j = k$, (2) will be violated.

Assume the contrary that $\mathcal{S}_j \geq 2\overline{S}$ and $\mathcal{L}_k \geq 2\overline{L}$. It suffices to prove that $\overline{C}_i > 2$ for all $i \in \{1, \ldots, M\}$, which implies a contradiction of $\sum_{i=1}^{M} \overline{C}_i > 2M$.

For all $i \in \{1, \ldots, M\}$, by Algorithm $\mathcal{SE}$, we have $\mathcal{L}_i < \overline{L} \Rightarrow \mathcal{S}_i \geq 2\overline{S}$, and $\mathcal{S}_i < \overline{S} \Rightarrow \mathcal{L}_i \geq 2\overline{L}$; and by (2), we have $\mathcal{L}_i = \overline{L} \Rightarrow \mathcal{S}_i > \overline{S}$, and $\mathcal{S}_i = \overline{S} \Rightarrow \mathcal{L}_i > \overline{L}$. The C-value for each of these cases is greater than two. We have two more cases: 1) $\mathcal{L}_i > \overline{L}$, and 2) $\mathcal{S}_i > \overline{S}$. Both cases lead to $\overline{C}_i > 2$. Consider the first case. If $\mathcal{L}_i > \overline{L}$ and $\overline{C}_i \leq 2$, then $\mathcal{S}_i < \overline{S}$, which implies $\mathcal{L}_i \geq 2\overline{L}$ as stated above and, therefore, $\overline{C}_i > 2$. This is a contradiction, and hence, $\mathcal{L}_i > \overline{L}$ implies $\overline{C}_i > 2$. Similarly, $\mathcal{S}_i > \overline{S}$ implies $\overline{C}_i > 2$. □

Lemma 1 shows that the if-conditions of Steps 3.3, 3.4, and 3.5 are complete. For the case in which $\mathcal{S}_j < 2\overline{S}$ and $\mathcal{L}_k < 2\overline{L}$, Step 3.3.1 swaps the content of the servers, and Step 3.3.2 places the new document into one of them. We argue that the loads and storage spaces of these two servers are bounded properly. Before swapping, we have $\mathcal{L}_j < \overline{L}$ and $\mathcal{S}_j < 2\overline{S}$. By (2), $(p_s^j - 1)\overline{S} < \mathcal{S}_j < 2\overline{S}$. It implies that $p_s^j < 3$, and by (1), we have $p_l^j \geq 3$. Similarly, we have $p_l^k < 3$ and $p_s^k \geq 3$. After swapping, both servers are available for the new document. Hence, after placing the new document in Step 3.3.2, their loads and storage spaces are bounded as required. The cost of reallocation for this case is $3\overline{S}$.

Lemma 2. *If Step 3.4.1 is executed, it outputs $(\mathcal{L}_i, \mathcal{S}_i)$ such that $\mathcal{S}_i + \mathcal{S}_j < 2\overline{S}$.*

Proof. We argue $\mathcal{S}_i < \overline{S}$ just after Step 3.4.1. Assume the contrary that $\mathcal{S}_i \geq \overline{S}$. According to Algorithm $\mathcal{SE}$, all servers have storage space $\overline{S}$. Then, $\mathcal{S}_j = \overline{S}$. Together with the fact that $\mathcal{L}_j < \overline{L}$, (2) is violated. Therefore, $\mathcal{S}_i < \overline{S}$. Let $\mathcal{S}_i = \delta\overline{S}$, for some $\delta \in (0, 1)$.

Divide the servers into three types. For type-1 servers, their loads are less than $\overline{L}$. For type-2 servers, their loads are at least $\overline{L}$, but less than $2\overline{L}$. A type-3 server has load at least $2\overline{L}$. By Algorithm $\mathcal{SE}$, $\mathcal{S}_i$ and $\mathcal{S}_j$ are the smallest values among all the choices, respectively. Hence, we have different lower bounds of storage spaces for the servers of different types as follows: For type-1 servers, their storage spaces are at least $\mathcal{S}_j$. As $\mathcal{L}_k \geq 2\overline{L}$, by Algorithm $\mathcal{SE}$, for all $r \in \{1, \ldots, M\}$, we have $\mathcal{L}_r \geq 2\overline{L}$ or $\mathcal{S}_r \geq \overline{S}$. Hence, a type-2 server has storage space at least $\overline{S}$. For type-3 servers, their storage spaces are at least $\delta\overline{S}$, as this is the lowest storage space among all servers.

Assume for contradiction that $\mathcal{S}_i + \mathcal{S}_j \geq 2\overline{S}$. Then, $\mathcal{S}_j \geq (2 - \delta)\overline{S}$. Let M_1, M_2, and M_3 be the number of servers of the three types, respectively. Consider the total storage space. We have $(M_1 + M_2 + M_3)\overline{S} \geq (M_1(2 - \delta) + M_2 + M_3\delta)\overline{S}$. It implies $M_3 \geq M_1$. Consider the total C-value. We have $2(M_1 + M_2 + M_3) > (2 - \delta)M_1 + 2M_2 + (2 + \delta)M_3$. Both sides cannot equal, as the loads of the type-1 servers are not taken into account in the right-hand side, and the existence of the jth server guarantees that the set of type-1 servers is nonempty. Thus, we have $M_1 > M_3$, which is a contradiction. □

For the case in which the condition in Step 3.4 is true, Step 3.4.1 finds another server, the ith server, with

minimum storage space, and Step 3.4.2 reallocates a minimal subset R of documents, which load is at least $\min(\mathcal{L}_i, L, l)$, from the ith server to the jth server. (Recall that l is the load of the new document.) After document reallocation, the ith server has room for the new document, and the resulting load is bounded by $p_l^j L$. The storage space is decreased and bounded by $p_s^j S$, obviously.

Consider the jth server. The size of R is at most $\mathcal{S}_i$, and the resulting $\mathcal{S}_j$ is still less than $2\overline{S} \le p_s^j \overline{S}$ by Lemma 2. If we can prove that the load of R is at most $\min(\mathcal{L}_i, 2\max(L, l))$, we can conclude that the final $\mathcal{L}_j$ is less than $\overline{L} + \min(\mathcal{L}_i, 2\max(L, l)) \le 3L$, which is then less than $p_l^j L$, where L stores its postplacement value. This result is because [$\mathcal{L}_j < \overline{L}$ and $\mathcal{S}_j < 2\overline{S}$] implies $p_s^j < 3$ by (2) and further gives $p_l^j > 3$ by (1).

The cost of reallocation is bounded by $\mathcal{S}_i$, which is less than $\overline{S}$, as discussed in the proof of Lemma 2. We now prove that the load of R is bounded by $\min(\mathcal{L}_i, 2\max(L, l))$. If $\mathcal{L}_i = \min(\mathcal{L}_i, L, l)$, R is the set of all documents in the ith server, and the claim is true. Consider the case $l = \min(\mathcal{L}_i, L, l)$. Searching for an optimal subset of documents such that its total load is at least and closest to l is NP-hard. Therefore, we use a heuristic to perform a linear search for a minimal contiguous subset R, which total load is just at least l. Then, the load of R is at most $\min(\mathcal{L}_i, \max(L, 2l))$. ($L$ is for the case that l is less than the load of the largest load document in the ith server, and $2l$ is for the opposite case.) Hence, the claim is true. For the case $L = \min(\mathcal{L}_i, L, l)$, by similar arguments, the load of R is at most $\min(\mathcal{L}_i, 2L)$.

On the other hand, the reallocation cost is at most $\min(\mathcal{S}_i, 2S, 2s) \le \mathcal{S}_i < \overline{S}$ if Step 3.5.1 is executed. Combining all cases, the reallocation cost is less than $3\overline{S}$.

Theorem 1. *After placement of an input document, we have $\mathcal{L}_j < p_l^j L$ and $\mathcal{S}_j < p_s^j S$, for all $j \in \{1, \ldots, M\}$, and the cost of document reallocation is less than $3\overline{S}$.*

The reallocation cost is a concern in the performance of the algorithms because it will be transformed into time cost. High reallocation cost can incur high response time and low throughput of the system and is certainly unfavorable. In our algorithm, the cost $3\overline{S}$ is $\frac{3}{M}$ of the total storage space. We believe that it is acceptable especially for large M. On the other hand, by document reallocation, the algorithm can handle heterogeneous servers and give an observable improvement on the result by Bilò et al. [2] in almost all the range of the load-storage space trade-off for homogeneous systems (Section 4).

4 A Better Trade-Off for Homogeneous Servers

Consider the best existing online algorithm for bicriteria scheduling [2], which is a (parametric) $(\frac{2M-k}{M-k+1}, \frac{M+k-1}{k})$-competitive algorithm, where $k \in \{1, \ldots, M\}$. It is called Algorithm $A(k)$ and is designed for homogeneous server systems without using document reallocation. 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 they satisfy $\frac{1}{t_l - 1} + \frac{1}{t_s - 1} = 1 + \frac{2}{M-1}$. We call this equation Curve C_{Ak}. In the curve, there are obviously $M - 1$ gaps separated by the M discrete pairs of Algorithm $A(k)$ (Fig. 1). Clearly, the points in the gaps cannot be used by Algorithm $A(k)$.

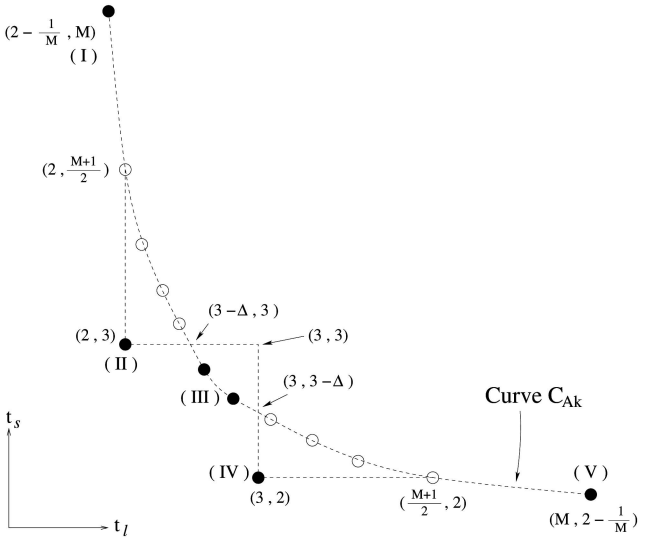


Fig. 1. A comparison between $A(k)$ and $HETER$ (not to scale).

The interaction point between Curve C_{Ak} and the horizontal line $t_s = 3$ is $(3 - \Delta, 3)$ and that between B and the vertical line $t_l = 3$ is $(3, 3 - \Delta)$, where $\Delta = \frac{8}{M+3}$. It implies that for each (t_l, t_s), from $(2, \frac{M+1}{2})$ to $(3 - \Delta, 3)$ and from $(3, 3 - \Delta)$ to $(\frac{M+1}{2}, 2)$, along Curve C_{Ak}, Algorithm $HETER$ outperforms[1] $A(k)$. The reallocation cost is no more than $\overline{S}$ when $HETER$ is applied in homogeneous servers, as the condition in Step 3.3 is never true, and the cost $\overline{S}$ follows from any one of Steps 3.4 and 3.5. We combine the advantages from each algorithm and form a new one. Fig. 1 shows the new set of upper bound pairs.

In the figure, the resulting upper bound pairs are shown by the solid dots. The white dots on Curve C_{Ak} are not used, as better solutions are (2, 3) and (3, 2). As a result, there are five portions for (t_l, t_s) as labeled in the figure. Portions (II) and (IV) are the two points from Algorithm $HETER$. Portions (I) and (V) are the extreme cases from Algorithm $A(k)$. As the little improvement of an additive term $\frac{1}{M}$ in one parameter does not justify the large cost of a factor $\frac{M}{3}$ on another one, these two pairs of values will probably not be used in practice.[2] For portion (III), directly from the boundary condition of t_l, we have $\frac{M}{2} - \frac{1}{2} < k < \frac{M}{2} + \frac{3}{2}$. Hence, if M is even, there are two discrete points $(\frac{3M}{M+2}, \frac{3M-2}{M})$ and $(\frac{3M-2}{M}, \frac{3M}{M+2})$. If M is odd, there is only a

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

2. Theoretically, they remain the best pairs, as there is no existing solution that outperforms them.

point $(\frac{3M-1}{M+1}, \frac{3M-1}{M+1})$. As shown in the figure, $HETER$ does not outperform this/these point(s). Possible reasons are that $HETER$ is basically designed for heterogeneous servers and that the bounds are independent of M. (Algorithm $A(k)$ is for homogeneous servers with parameter M.)

In this section, we emphasize on homogeneous servers. With the help of document reallocation, we form a continuous set of realizable upper bound pairs along Curve C_{Ak} of the middle portion, for all $M \geq 18$. Formally, given any (t_l, t_s) such that $3 - \Delta < t_l < 3$ and $\frac{1}{t_l-1} + \frac{1}{t_s-1} \geq 1 + \frac{2}{M-1}$, we give an algorithm to perform online placement such that $\mathcal{L}_j < t_l L$ and $\mathcal{S}_j < t_s S$, for all $j \in \{1, \ldots, M\}$.

Within this range of t_l in Curve C_{Ak}, for each (t_l, t_s) not used by Algorithm $A(k)$, there exists an integer k laying between $\lfloor \frac{M}{2} - \frac{1}{2} \rfloor$ and $\lceil \frac{M}{2} + \frac{3}{2} \rceil$ such that

$$\begin{aligned} \frac{2M-k}{M-k+1} < t_l < \frac{2M-k-1}{M-k} \text{ and} \\ \frac{M+k}{k+1} < t_s < \frac{M+k-1}{k}, \end{aligned} \tag{3}$$

as $(\frac{2M-k}{M-k+1}, \frac{M+k-1}{k})$ and $(\frac{2M-k-1}{M-k}, \frac{M+k}{k+1})$ are the two nearest points surrounding (t_l, t_s) on B.

Upon the arrival of a document of load l and size s, if we can find a server with load 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. Thus, the resulting load is at most $t_l L$, and storage space is at most $t_s S$, where $\overline{L}$ and $\overline{S}$ store the preplacement values, and L and S store the postplacement ones. It is because the new load is at most $\frac{M}{M-1}(t_l - 1)\overline{L} + l = \frac{M}{M-1}(t_l - 1)(\overline{L}' - \frac{l}{M}) + l = \frac{M}{M-1}(t_l - 1)\overline{L}' + (1 - \frac{t_l-1}{M-1})l \leq \frac{M}{M-1}(t_l - 1)L + (1 - \frac{t_l-1}{M-1})L = t_l L$, where $\overline{L}'$ is the new average load of the server. A similar argument can be made for the storage space.

Let P be the set of servers with loads more than $\frac{M}{M-1}(t_l - 1)\overline{L}$, and Q be the set of servers with storage spaces more than $\frac{M}{M-1}(t_s - 1)\overline{S}$. They are stored in separate B^0-trees. 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 $P \cap Q \neq \emptyset$, 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 $|Q| \leq \frac{M-1}{t_s-1} - 1$, and it also implies the existence of one server outside $P \cup Q$.

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 $\lfloor \frac{M-1}{t_l-1} \rfloor + \lfloor \frac{M-1}{t_s-1} \rfloor = M$. Because there is no available server for the new document, we apply document reallocation to vacate a server, and Theorem 2 below shows this possibility.

Theorem 2. *There exists an algorithm such that for all $M \geq 18$, 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 less than $t_l\overline{L}$ and $t_s\overline{S}$, respectively.*

The idea of the algorithm can be illustrated by the fact that if almost all the load and storage space of the whole system are in P and Q, respectively, we can find a server in P with little storage space and a server in Q with little load. Then, it will be safe to move the documents in the server of P to the server of Q with little reallocation cost. In practice, we simply take away the minimal number of documents to avoid the excessive reallocation cost.

Proof. We first claim that

$$|P| = \left\lfloor \frac{M-1}{t_l - 1} \right\rfloor \text{ and } |Q| = \left\lfloor \frac{M-1}{t_s - 1} \right\rfloor. \tag{4}$$

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 $|P| \geq \lfloor \frac{M-1}{t_l-1} \rfloor + 1 > \frac{M-1}{t_l-1}$. 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 account, we have

$$\begin{aligned} |P| < \frac{M\overline{L} - \delta_Q\overline{L}}{\frac{M}{M-1}(t_l - 1)\overline{L}} = \frac{(M - \delta_Q)(M-1)}{M(t_l - 1)} \text{ and} \\ |Q| < \frac{M\overline{S} - \delta_P\overline{S}}{\frac{M}{M-1}(t_s - 1)\overline{S}} < \frac{(M - \delta_P)(M-1)}{M(t_s - 1)}. \end{aligned} \tag{5}$$

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 (5), 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. Hence, the claim is true, and it implies that $\delta_Q < 1 + \frac{t_l-1}{t_s-1} < 1 + \frac{2}{2-\Delta} \leq 3$, for $M \geq 5$, and similarly, $\delta_P < 3$.

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 X. In other words, X is the smallest load server with storage space no more than $\frac{\delta_P\overline{S}}{q}$. Obviously, X exists and the loop terminates. Similarly, find the smallest storage space server Y in Q with load no more than $\frac{\delta_Q\overline{L}}{q}$. Without loss of generality, assume that Y is the c'th smallest storage space server in Q.

Before document reallocation, 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 searching algorithm is $O(q \log M)$, which is $O(\log M)$, as q will be set as a constant.

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

$$\leq \left(\frac{\delta_Q}{q} + \frac{M-\delta_Q-(c-1)\frac{M}{M-1}(t_l-1)}{\left\lfloor\frac{M-1}{t_l-1}\right\rfloor-(c-1)}\right)\overline{L}$$

by (4)

$$\leq \left(\frac{\delta_Q}{q} + \frac{M-\delta_Q-(q-1)\frac{M}{M-1}(t_l-1)}{\left\lfloor\frac{M-1}{t_l-1}\right\rfloor-(q-1)}\right)\overline{L}$$

since $\left\lfloor\frac{M-1}{t_l-1}\right\rfloor < \frac{M-\delta_Q}{\frac{M}{M-1}(t_l-1)}$ and $c \leq q$

$$< \left(\frac{\delta_Q}{4} + \frac{M-\delta_Q-(3)\frac{M}{M-k+1}}{M-k-3}\right)\overline{L}$$

by (3), and setting $q = 4$

$$\leq \left(\frac{3}{4} + \frac{M-3-\frac{3M}{M-k+1}}{M-k-3}\right)\overline{L}$$

by $\delta_Q < 3$ and $4 \leq M-k-3$ (as $M \geq 18$)

$$< \left(1 + \frac{M-1}{M-k+1}\right)\overline{L}$$

by $k < \left\lceil\frac{M}{2}+\frac{3}{2}\right\rceil$

$$< t_l\overline{L},$$

by (3)

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

Below is the algorithm mentioned in Theorem 2.

Algorithm $Find$:
1. Let S_P be the total storage space in P, and L_Q be the total load in Q;
2. $q := \min(|P|, |Q|, 4)$; $c := 0$; $c' := 0$;
3. Loop $c := c+1$; $X :=$ the cth smallest load server in P;
   Quit if the storage space of X is no more than $\frac{S_P}{q}$;
4. Loop $c' := c'+1$;
   $Y :=$ the c'th smallest storage space server in Q;
   Quit if the load of Y is no more than $\frac{L_Q}{q}$;
5. Return X and Y;

Because the theorem bridges the gaps between the discrete points, we now redefine portion (III) by removing the constraint of k and give an algorithm $SMOOTH$ for this new portion as follows: Steps 1 and 2 first check if document reallocation can be avoided. For the discrete points used by Algorithm $A(k)$, placement of d must be done in Step 2.1 without reallocation. Step 3 focuses on the points that cannot be handled by $A(k)$. We use $Find$ to find X and Y from P and Q, respectively. Move all the documents in X to Y, while Theorem 2 guarantees that Y's bounds will not be violated. Then, X is available for d. Note that, in order to reduce the reallocation cost, only minimal documents in X will be reallocated in practice.

Algorithm $SMOOTH(t_l, t_s)$: // $\frac{1}{t_l-1}+\frac{1}{t_s-1} = 1+\frac{2}{M-1}$ and $3-\Delta < t_l < 3$.

Upon the arrival of a document d
1. Perform $\mathcal{SE}^*$ 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 $Find$ and get 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 $\overline{S}$;

4.1 Remarks

In our algorithm, $M \geq 18$ when $q = 4$, as $M-k-3 \geq 4$ is true in proving of $\mathcal{L}_j < t_l\overline{L}$. If we choose $q = 5$, then $M \geq 20$, and $M \geq 22$ when $q = 6$, etc. Recalling that the reallocation cost is bounded by $\frac{3\overline{S}}{q}$, for any M, we must use the greatest q in order to reduce the reallocation cost. For the case that M is a parameter during the system design, choosing the value of M is to decide a trade-off between the reallocation cost and the other parameters such as the maintenance cost incurred. (If M increases, the maintenance cost increases, but the reallocation cost $\frac{\delta_P\overline{S}}{q}$ monotonically decreases.) As $M \to \infty$, the interaction points $(3-\Delta, 3)$ and $(3, 3-\Delta)$ converge at (3, 3), and the extreme points $(2-\frac{1}{M}, M)$ and $(M, 2-\frac{1}{M})$ converge at $(2, \infty)$ and $(\infty, 2)$, respectively. It means that, asymptotically, Algorithm $HETER$ outperforms $A(k)$ everywhere.

5 A Special Case for Homogeneous Servers

In this section, our aim is to bound the load, the storage space, and the capacity index of each server by $3L$, $3S$, and 4, respectively, after each document placement.

Recall that the two points (2, 3) and (3, 2) are asymptotically the best results at present. The result in this section implies that if we sacrifice one parameter and set the upper bounds of load and storage space to $3L$ and $3S$, respectively, then we gain in the capacity index. The capacity index measures the integrated effect of load and storage space on each server. Directly from (1), the trivial upper bound for a capacity index is 5 and will be improved to 4 in this section. It is important to note that with this improvement on the capacity index, if one parameter in a server is near its worst case $(3L/3S)$, the other one must be close to its trivial lower bound (S/L). For example, algorithm $HETER$ allows a server's load to be close to $2L$ and its storage space to be close to $3S$, simultaneously. Algorithm $CAPACITY$ allows the load to grow close to $3L$, but if the load really does, then the storage space will be bounded near S, regardless of its upper bound $3S$. The improvement in capacity index also gives hope that both parameters could be very close to twice of their trivial lower bounds simultaneously, which is compatible with the asymptotic latest known upper bound of 1.9201 [7] and the lower bound of 1.88 [12] for balancing a single

parameter. The price for it is the cost $1.5\overline{S}$ in reallocating documents. It could be improved to $(1+\alpha)\overline{S}$ in Section 5.1, where α is an arbitrary positive constant less than 0.5, and the time needed would increase by a constant factor $\frac{1}{\alpha}$. 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 avoid any document reallocation. For small M, the average storage space is large, and this trivial approach is a better choice. However, when M is large or the cost of document reallocation is minor, an $O(\log M)$-time algorithm $CAPACITY$ will be given. Its idea is given 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 $\overline{S}$, respectively, the algorithm will find it and place d into it. After placement, the load will clearly be no more than $2L$, the storage space no more than $2S$, and the capacity index no more than four. Suppose no such server exists in the system. That is, $\forall i \in \{1,\ldots,M\}$, $\mathcal{L}_i > \overline{L}$ or $\mathcal{S}_i > \overline{S}$. As the sum of all C-values is $2M$, there must be a server, say, the ith server, which C-value is at most two. Therefore, we have [$\mathcal{L}_i < \overline{L}$ and $\overline{S} < \mathcal{S}_i < 2\overline{S}$] or [$\mathcal{S}_i < \overline{S}$ and $\overline{L} < \mathcal{L}_i < 2\overline{L}$]. The former case is tackled by Phase A and the latter by Phase B. Since they are symmetric, we only discuss the first case. Without loss of generality, we assume the ith server has the smallest storage space, i.e., if another server has a smaller storage space and its load is less than $\overline{L}$, then its C-value is greater than two.

Algorithm $CAPACITY$:
Upon the arrival of a document d with load l and size s
1. Perform $\mathcal{SE}^*$ on T_2 with input $(\overline{L},\overline{S})$ 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 Phase A; (as defined later)
3.2 If d cannot be successfully placed in Phase A
3.2.1 Perform Phase B;
4. Update $\overline{L}$ and $\overline{S}$;

For simplicity, two procedures, namely, $PLACE(D,\alpha)$ and $GET(x)$, will be used in Phase A. $PLACE(D,\alpha)$ performs the following steps: If $\overline{C}_\alpha \leq 4 - \overline{C}_D$, where $\overline{C}_D$ is the C-value contributed by a set D of documents, then place D into the αth server; else, quit the current phase. The procedure $GET(x)$ is given below.

Procedure $GET(x)$:
G1. Perform $\mathcal{SE}$ on T_2 with input x and get output;
G2. If output is false
G2.1 Quit the current phase;
G3. If output is $(\mathcal{L}_\alpha,\mathcal{S}_\alpha)$
G3.1 If $\mathcal{S}_\alpha \geq 2\overline{S}$ Quit the current phase;
G3.2 If $\overline{C}_\alpha \leq 2$
G3.2.1 Place d into the αth server;
Quit the current phase;
G3.3 If $\overline{C}_\alpha > 2$ return α;

Phase A is given below:

A1. Perform $GET(\overline{L})$ and get output j;
A2. Perform $GET(\mathcal{L}_j)$ and get output k;
A3. Perform $GET(\mathcal{L}_k)$ and get output q;
A4. Partition the jth server in the following way: If there is a document with size more than $\overline{S}$, store it in a bucket and the rest of documents in another bucket;
Else, store all documents in at most three buckets such that each bucket has storage space at most $\overline{S}$; Let B and B' be the buckets which C-values c and c' are the largest and the second largest ones no more than 1.5, respectively;
A5. If B exists
A5.1 If $\overline{C}_k \leq 4 - c$
A5.1.1 Pick the bucket B; $PLACE(B,k)$;
A5.1.2 If the new $\overline{C}_j > 2$ (*i.e.*, the old $\overline{C}_j - c > 2$) and B' exists
A5.1.2.1 Pick the bucket B'; $PLACE(B',q)$;
A5.2 Else
A5.2.1 Perform $GET(\mathcal{L}_k - (2-c)\overline{L})$ and get output p;
A5.2.2 Pick the bucket B; $PLACE(B,p)$;
A6. Place d into the jth server;

In procedure $GET(x)$, Step G1 searches for a server with the smallest storage space and load bounded by x. GET will quit from the current phase without placement if no such server exists, or it exists with storage space at least $2\overline{S}$. If Step G1 outputs a server with C-value at most two, GET places d into this server and quits the current phase; it will return this server to Phase A if its C-value is greater than two.

In Phase A, Steps A1 to A3 output three servers, namely, jth, kth, and qth servers, if none of them can accept d. (Otherwise, it is done.) Then, the remaining task of Phase A is to place d into the jth server after reallocating some of its documents.

The documents are partitioned into at most three buckets based on the criteria in Step A4, and at most two of them, namely, B and B' if they exist, will be reallocated. If B does not exist, the jth server has only one document and d is directly placed in the jth server. Suppose B exists. If the kth server can accept B without violating the limit on C-value, then reallocation of B will be done in Step A5.1.1, and if the C-value of the jth server is still higher than 2, and B' exists, then B' will be reallocated to the qth server in Step A5.1.2.1. For the case that B' does not exist, there is only one "big" document left in the jth server after B has been reallocated, and therefore, it can accept d. If the kth server cannot accept B due to the violation on C-value, B will be moved to another server returned by GET in Step A5.2.1. For both cases, the resulting jth server can accept d.

In practice, if Step A5.1.2.1 is executed, then Step A5.1.1 does not reallocate B before B' is successfully reallocated. Phase A is designed for the case [$\mathcal{L}_i < \overline{L}$ and $\overline{S} < \mathcal{S}_i < 2\overline{S}$]. If this condition is not true, GET and $PLACE$ may quit the current phase without any placement and reallocation. Phase B will then perform the placement.

Phase B can be obtained by the following steps: 1) Swap the positions of load and storage space, $\overline{L}$ and $\overline{S}$, and $\mathcal{L}_*$ and $\mathcal{S}_*$ in Phase A, and $PLACE$ and GET. 2) Use T_3 instead of T_2. 3) Skip the testing in $PLACE$ and Step G3.1

of GET, as Phase B must succeed in placing d after Phase A fails to do so.

Theorem 3. *Suppose $\mathcal{L}_i < \overline{L}$ and $\overline{S} < \mathcal{S}_i < 2\overline{S}$. The document d can be placed into a server in Phase A, and after the completion of the algorithm, each server has load less than $3L$, storage space less than $3S$, and capacity index no more than four. The reallocation cost is no more than $1.5\overline{S}$.*

Proof. Suppose d is placed in Step G3.2.1 in GET. By Steps G3.1 and G3.2, the resulting $\mathcal{S}_\alpha$ is less than $3S$, and C_α is no more than four. As for load, if GET is invoked in Step A1, before placement, $\mathcal{L}_\alpha < \overline{L}$. Similarly, we have $\mathcal{L}_\alpha < \mathcal{L}_j < \overline{L}$ if GET is called in Step A2, and $\mathcal{L}_\alpha < \mathcal{L}_k < \mathcal{L}_j < \overline{L}$ if GET is called in Step A3. Hence, $\mathcal{L}_\alpha < 2L$ after placing d.

It now suffices to prove that if d is not placed in Steps A1 to A3 and A5.2.1, then it will be successfully placed into the jth server without violating the bounds of loads, storage spaces, and capacity indices of all the servers.

Since $\mathcal{L}_i < \overline{L}$, Step G1 in $GET(\overline{L})$ does not return false. By algorithm $\mathcal{SE}$, the only reason why $GET(\overline{L})$ does not return i is $\mathcal{S}_j < \mathcal{S}_i$. As $\mathcal{L}_j < \overline{L}$, by the choice of i, we have $\overline{C}_j > 2$. As $\overline{S} < \mathcal{S}_i < 2\overline{S}$, we have $\mathcal{S}_j < 2\overline{S}$, and therefore, $GET(\overline{L})$ will not quit Phase A but will return j. Considering $\overline{C}_i < 2$, we have $\mathcal{L}_i < \mathcal{L}_j < \overline{L}$ and $\overline{S} < \mathcal{S}_j < \mathcal{S}_i < 2\overline{S}$. Similarly, by considering the fact that $GET(\mathcal{L}_j)$ does not return i but k, and $GET(\overline{L})$ does not return k but j, we have $\overline{C}_k > 2$, $\mathcal{L}_i < \mathcal{L}_k < \mathcal{L}_j < \overline{L}$ and $\overline{S} < \mathcal{S}_j < \mathcal{S}_k < \mathcal{S}_i < 2\overline{S}$. By similar arguments, we have $\overline{C}_q > 2$, $\mathcal{L}_i < \mathcal{L}_q < \mathcal{L}_k < \mathcal{L}_j < \overline{L}$, and $\overline{S} < \mathcal{S}_j < \mathcal{S}_k < \mathcal{S}_q < \mathcal{S}_i < 2\overline{S}$. Hence, Steps A1 to A3 do not quit Phase A but return j, k, and q, respectively.

If B does not exist in Step A4, here is only one document in the jth server. Step A6 places d into the server without violating the bounds. Suppose B exists. If $\overline{C}_k \le 4 - c$, Step A5.1.1 reallocates B to the kth server. The resulting $\overline{C}_k$ is clearly no more than four. Since $\mathcal{L}_j < \overline{L}$ and $\mathcal{L}_k < \overline{L}$ before reallocation, we have $\mathcal{L}_k < 2\overline{L}$ after reallocation. If B has more than one document, its storage space is bounded by $\overline{S}$; else, it is bounded by S. Hence, the resulting $\mathcal{S}_k < 2\overline{S} + S \le 3S$. The same arguments apply to the qth server, if Step A5.1.2.1 is executed. After Step A5.1.1, if $\overline{C}_j > 2$, and B' does not exist, then there is only one document left in the jth server, and this server can accept d. If B' exists and needs to be reallocated, there is at most one bucket left inside the jth server after the second reallocation. Then, C_j becomes less than two, and the jth server can accept d.

Consider $\overline{C}_k > 4 - c$. Since $\mathcal{S}_k < 2\overline{S}$, we have $\mathcal{L}_k > (2-c)\overline{L}$. Recalling that $\mathcal{S}_k < \mathcal{S}_i < 2\overline{S}$, $\overline{L} > \mathcal{L}_k > \mathcal{L}_i$, and $\overline{C}_k - (2-c) > 2 > \overline{C}_i$, we have $\mathcal{L}_i < \mathcal{L}_k - (2-c)\overline{L}$. Hence, the existence of the ith server guarantees that GET in Step A5.2.1 can return the pth server such that $\mathcal{L}_p < \mathcal{L}_k - (2-c)\overline{L}$. If $p = i$, d can be placed in the ith server directly. If $p \ne i$, then we have $\mathcal{L}_i < \mathcal{L}_p < \mathcal{L}_k - (2-c)\overline{L} < \mathcal{L}_k < \overline{L}$ and $\mathcal{S}_k < \mathcal{S}_p < \mathcal{S}_i < 2\overline{S}$. Then, $\overline{C}_p = \frac{\mathcal{L}_p}{\overline{L}} + \frac{\mathcal{S}_p}{\overline{S}} < \frac{\mathcal{L}_k - (2-c)\overline{L}}{\overline{L}} + 2 < \frac{\mathcal{L}_k}{\overline{L}} + c < 1 + c$. Hence, B is successfully reallocated to the pth server, because $\overline{C}_p$ will be less than $1 + c + c \le 4$ and $\mathcal{L}_p$ less than $\overline{L} + \overline{L} = 2\overline{L} \le 2L$ and $\mathcal{S}_p$ less than $2\overline{S} + \overline{S} = 3\overline{S} \le 3S$, after B is reallocated to the pth server. Recalling $\overline{C}_k > 4 - c$ and $\overline{C}_k = \frac{\mathcal{L}_k}{\overline{L}} + \frac{\mathcal{S}_k}{\overline{S}} < 3$, we have $c > 1$. As $\overline{C}_j < 3$ initially, $\overline{C}_j$ is less than two after reallocating B, and therefore, the jth server can now accept d.

Consider the reallocation cost. If only B is reallocated, the cost is bounded by $1.5\overline{S}$, as $c \le 1.5$. If B' exists and will be reallocated, c must be less than one; otherwise, the new $\overline{C}_j$ will not be more than two, and Steps A6.3.1 and A6.3.2 will not be executed. Hence, $c' \le c < 1$. There should be a third bucket, because $\overline{C}_j$ is still more than two. We call this bucket B^2. After B is taken away, the C-value contributed by B' and B^2 is more than two. It means that the C-value of B^2 is more than one, and by Step A4, it must be no less than 1.5. As the load of B^2 is less than $\overline{L}$, the storage space of B^2 is more than $\frac{\overline{S}}{2}$. It implies that the total storage space of B and B' is less than $1.5\overline{S}$. For Phase B, the storage space of jth server is less than $\overline{S}$, and hence, the reallocation cost is bounded by $1.5\overline{S}$. □

5.1 Improving the Reallocation Cost

The upper bound, $1.5\overline{S}$, of the reallocation cost can be reduced to $(1+\alpha)\overline{S}$, for any positive constant $\alpha \in (0, \frac{1}{2})$. We first change the upper bound for c to $2 - \alpha$. Consider that there is no document with size greater than $\overline{S}$. If both B and B' exist and will be reallocated, the C-value of the remaining documents is greater than $2 - \alpha$. The storage space of B^2 is more than $(1-\alpha)\overline{S}$, and the total storage space of B and B' will then be less than $(1+\alpha)\overline{S}$. This is done in Steps A5.1.1 and A5.1.2.1. If only B is reallocated, its storage space is less than $\overline{S}$, and its C-value is less than $2 - \alpha$. For the case of a document with size greater than $\overline{S}$, let B be the bucket with storage space less than $(1+\alpha)\overline{S}$ and the smallest C-value. Its C-value is obviously less than $2 - \alpha$. Note that if the storage space is greater than $\overline{S}$, B contains only one document and will not violate the bound of storage space upon reallocation. Combining all cases, it remains to find a server to accept a bucket with C-value less than $2 - \alpha$.

Recall in the proof of Theorem 3 that we have $\overline{C}_p < 1 + c$, and, therefore, it may not be possible to reallocate B to the pth server, as well as the kth one, for $c \le 2 - \alpha$. We now modify Phase A and argue that we can find a server for B. If the condition in Step A6.1 is false, as mentioned, $\mathcal{L}_i < \mathcal{L}_k - (2-c)\overline{L} < \mathcal{L}_k - \alpha\overline{L}$. Similarly, if $\overline{C}_p > 4 - c$, we have $\mathcal{L}_i < \mathcal{L}_p - (2-c)\overline{L}$, where $\mathcal{L}_p < \mathcal{L}_k - (2-c)\overline{L}$. Hence, $\mathcal{L}_i < \mathcal{L}_k - 2(2-c)\overline{L} < \mathcal{L}_k - 2\alpha\overline{L}$. Note that initially, $\mathcal{L}_i < \mathcal{L}_k - \alpha\overline{L}$, and after one execution of Step A5.2.1, if the pth server cannot accept B, we have $\mathcal{L}_i < \mathcal{L}_k - 2\alpha\overline{L}$. Repeat Step A5.2.1 (with a modification of the subscripts) for $\lceil \frac{1}{\alpha} \rceil$ times; if there is no suitable server found, then $\mathcal{L}_i < \mathcal{L}_k - \overline{L}$, which is a contradiction. Intuitively, the gap between $\mathcal{L}_i$ and $\mathcal{L}_k$ can be as large as $\overline{L}$, but the gap between $\mathcal{L}_i$ and $\mathcal{L}_p$ is at most $\overline{L} - (2-c)\overline{L} < (1-\alpha)\overline{L}$. It suffices to test at most $\lceil \frac{1}{\alpha} \rceil$ servers with loads between $\mathcal{L}_i$ and $\mathcal{L}_k$. Hence, the time needed is $O(\frac{1}{\alpha} \log M)$.

6 Conclusion

We give three results for balancing the loads and storage spaces among servers using document reallocation. In order to obtain the best trade-off, we need to equalize both sides of each inequality in (1). The system designer can then determine the trade-off between load and storage space for each server in the heterogeneous case or for the whole system in the homogeneous case.

For heterogeneous servers, the first algorithm guarantees that for all $j \in \{1, \ldots, M\}$, $\mathcal{L}_j < p_l^j L$ and $\mathcal{S}_j < p_s^j S$, where $[p_l^j \geq 2$ and $p_s^j \geq 3]$ or $[p_l^j \geq 3$ and $p_s^j \geq 2]$. We assume that the heterogeneity among servers can be translated into different choices of p_l^j and p_s^j, $j \in \{1, \ldots, M\}$. The reallocation cost is less than $3\overline{S}$.

For homogeneous servers, our second result is a combination of our first algorithm and Algorithm $A(k)$ in [2]. With document reallocation, it gives $\mathcal{L}_j \leq t_l L$ and $\mathcal{S}_j \leq t_l S$, for all $j \in \{1, \ldots, M\}$, where $(t_l, t_s) \in \{(2 - \frac{1}{M}, M), \ (M, 2 - \frac{1}{M})\} \cup \{(2,3), (3,2)\} \cup \{(t_l, t_s) | \frac{1}{t_l - 1} + \frac{1}{t_s - 1} \geq 1 + \frac{2}{M-1}\}$. Graphically, our contribution is to smooth the middle portion (Fig. 1). The document reallocation required is only $\frac{3\overline{S}}{q}$, where q is some integer at least four. Together with the points $(2 - \frac{1}{M}, M)$, $(M, 2 - \frac{1}{M})$, (2, 3), and (3, 2), the reallocation cost is less than $3\overline{S}$, which is dominated by the first result.

Our last algorithm gives $\mathcal{L}_j < 3L$, $\mathcal{S}_j < 3S$ and $C_j \leq 4$, for all $j \in \{1, \ldots, M\}$. It implies that in each server, at least one of load and storage space is no more than twice its trivial lower bound, although the upper bounds of load and storage space are not as tight as those of the first and second results. The reallocation cost is $(1 + \alpha)\overline{S}$, where α is any arbitrary positive constant less than $\frac{1}{2}$.

Our solutions are based on large values of M, which implies small values of $\overline{S}$, and hence, small reallocation costs. Further research is needed to reduce the reallocation cost for general values of M. Another direction may be to study different models of server heterogeneity.

References

[1] G.C. Amita, "Incremental Data Allocation and Reallocation in Distributed Database Systems," *Data Warehousing and Web Eng.*, pp. 137-160, 2002.

[2] V. Bilò, M. Flammini, and L. Moscardelli, "Pareto Approximations for the Bicriteria Scheduling Problem," *J. Parallel and Distributed Computing*, vol. 66, no. 3, pp. 393-402, 2006.

[3] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Compact, Adaptive Placement Schemes for Non-Uniform Requirements," *Proc. ACM Symp. Parallelism in Algorithms and Architectures (SPAA '02)*, Aug. 2002.

[4] 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.

[5] 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 (CLUSTER '01)*, pp. 274-281, Oct. 2001.

[6] 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.

[7] R. Fleischer and M. Wahl, "Online Scheduling Revisited," *J. Scheduling*, special issue on approximation algorithms for scheduling algorithms (Part 2), vol. 3, no. 6, pp. 343-353, 2000.

[8] E. Haddad, "Runtime Reallocation of Divisible Load under Processor Execution Deadlines," *Proc. Third Workshop Parallel and Distributed Real-Time Systems (WPDRTS '95)*, pp. 30-31, Apr. 1995.

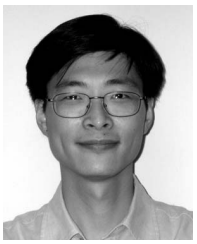
[9] H. Harada, Y. Ishikawa, A. Hori, H. Tezuka, S. Sumimoto, and T. Takahashi, "Dynamic Home Node Reallocation on Software Distributed Shared Memory," *Proc. Fourth Int'l Conf./ Exhibition on High Performance Computing in the Asia-Pacific Region (HPC-ASIA '00)*, vol. 1, pp. 158-163, May 2000.

[10] D.E. Knuth, *The Art of Computer Programming*, vol. 3. Addison-Wesley, 1973.

[11] A. Rasala, C. Stein, E. Torng, and P. Uthaisombut, "Existence Theorems, Lower Bounds and Algorithms for Scheduling to Meet Two Objectives," *Proc. 13th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA '02)*, pp. 723-731, 2002.

[12] J.F. Rudin III, "Improved Bounds for the Online Scheduling Problem," PhD dissertation, Univ. of Texas at Dallas, 2001.

[13] S.S.H. Tse, "Approximation Algorithms for Document Placement in Distributed Web Servers," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 6, pp. 489-496, June 2005.



Savio S.H. Tse received the BSc degree in physics and the PhD degree in computer science from the University of Hong Kong in 1988 and 1997, respectively. After graduation, he had been a visiting assistant professor and a guest lecturer at the University of Hong Kong for three years and a lecturer at the Hong Kong Polytechnic University for three years. He joined the Department of Computer Engineering, Bilkent University, in 2005. His research interests include parallel and distributed computing.