

WebWave: Globally Load Balanced Fully Distributed Caching of Hot Published Documents

Abdelsalam Heddaya*
heddaya@cs.bu.edu

Sulaiman Mirdad†
mirdad@cs.bu.edu

Computer Science Department
Boston University

Abstract

Document publication service over such a large network as the Internet challenges us to harness available server and network resources to meet fast growing demand. In this paper, we show that large-scale dynamic caching can be employed to globally minimize server idle time, and hence maximize the aggregate server throughput of the whole service. To be efficient, scalable and robust, a successful caching mechanism must have three properties: (1) maximize the global throughput of the system, (2) find cache copies without recourse to a directory service, or to a discovery protocol, and (3) be completely distributed in the sense of operating only on the basis of local information.

In this paper, we develop a precise definition, which we call tree load-balance (TLB), of what it means for a mechanism to satisfy these three goals. We present an algorithm that computes TLB off-line, and a distributed protocol that induces a load distribution that converges quickly to a TLB one. Both algorithms place cache copies of immutable documents, on the routing tree that connects the cached document's home server to its clients, thus enabling requests to stumble on cache copies en route to the home server.

Keywords: load balancing, caching, replication, document service, read-only files, packet filter, routing, Internet, World Wide Web.

1 Introduction

Current multimedia Internet information services provide users with the capability to transfer documents on a scale that is very large and fast growing. For

*Some of this work was done while this author was on sabbatical leave at Harvard University, 1995/96.

†This author's research is supported in part by a scholarship from the Saudi Cultural Mission to the U.S.A.

document services to keep up with quickly increasing demand, servers need to cooperate so that minimal capacity goes idle in one part of the network, when other parts have excess load. This must be done by creating cache copies of popular documents on lightly loaded nodes, and shifting requests from the heavily loaded servers to those nodes. But the effectiveness of such a system hinges on its ability to scale to many millions of server nodes.

Scalability¹ is determined by two factors: the overhead associated with implementing the protocol, and its reliance on centralized services. Most Current research [4, 5, 16, 25] assumes the existence of a cache directory service, that is queried when client requests are processed. The overhead associated with such a service limits the scalability of the caching system as a whole [17]. This paper attempts to answer the following question: can a document service achieve an optimal load distribution strictly through the use of local information? We give detailed evidence in the affirmative, assuming that the document service is allowed to integrate—to a limited extent—caching with routing. Specifically, we assume that routers can accept filters, supplied by cache servers, that identify requests that represent potential hits in the cache.

A significant amount of research has been conducted on the use of caching to improve the performance of file systems. Examples of such work include: the Network File System (NFS) [27], the Andrew File System (AFS) [20], Sprite [26], HARP [22], and xFS [12]. In addition, caching is used to improve the performance of large document delivery services such as USENET, FTP, and the World-wide Web [4, 5, 16, 25]. For example the Harvest cache [9], initially motivated by FTP mirroring, is now being increasingly used in national WWW caching systems around the world.

¹The rate at which the global throughput of the system grows as its total capacity increases.

To achieve load balance through file caching, requests have to be (re)directed to cache copies whose locations must therefore be determined. A common approach that has been adopted for load balancing over local area networks, performs this redirection by modifying the naming [21] or the routing [1] mechanisms to maintain an explicit directory of cache copies, from which to select suitable destinations for client requests. Frequent and pronounced changes in request patterns can force the locations, and number, of cache copies to be highly transient. The resulting high update rate of the cache directory means that it cannot be replicated efficiently on a large scale, thus turning the cache directory service itself into a scalability bottleneck, for caching systems that funnel all requests through it. Another strategy is to augment the client/server interaction protocol so that clients, or their proxies, proactively identify suitable cache copies using a special protocol, such as ICP [28]. This latter approach introduces additional complexity, message cost and extra round-trip message delays.

Our approach exploits the fact that routes from clients to a server form a *routing tree*, along which all document requests must flow. We observe that client requests are routed up the tree to the home server, and can be serviced by any node *en route* that happens to cache the desired document, without any need to lookup a cache directory, to redirect requests, or to probe the network to locate a copy. In *WebWave*, cache copies are created only when a parent node in the routing tree detects a less loaded child, to which it can shift some of its document service load by giving it a copy of one of its cached documents. An imbalance in the opposite direction causes a child to delete some of its cached documents, or to reduce the fraction of requests for these documents that it chooses to serve. In this paper, we concentrate on the problem of defining and computing the request rate that each node in the routing tree should handle in order for the tree as a whole to have maximum throughput. Choosing the particular documents to copy, or which copies to delete, is also discussed, but only briefly.

From the architectural point of view, a *WebWave* cache server needs to be able to insert a packet filter into the router associated with it, so that only document request packets that are highly likely to hit in the cache, are extracted from their normal path. Engler and Kaashoek [13] use run-time code generation techniques to dynamically download high performance packet filters into the kernel. Their measured overhead rivals that of hand crafted kernel code (a packet can be filtered in 1.51 microseconds). This demonstrates the feasibility, in principle, of our architectural requirement

for routers to accept such injected filtering code.

In summary, the main contributions of this paper include: formal definition of optimal *tree load balance* (TLB), an algorithm (*WebFold*) to compute it off-line, and a distributed protocol (*WebWave*) that induces a load distribution that converges to a TLB one. The remainder of the paper is organized as follows: Section 2 presents the diffusion method, and its convergence to Global Load Equality (*i.e.*, uniform load). Section 3 provides formal definitions of *Tree Load Balance* (TLB) and the underlying constraints. Section 4 presents *WebFold*, an off-line algorithm that finds a TLB assignment, and proves its correctness. Section 5 describes *WebWave* a fully distributed caching algorithm, and shows preliminary simulation evidence of its optimality (convergence to TLB). Finally, Section 7 concludes the paper and presents future work.

2 Load Diffusion

Our underlying model is similar to the dynamic load balancing model in Cybenko [11], and Bertsekas and Tsitsiklis [3]. In particular, *WebWave* consists of a set of caching servers cooperating to service client requests. The objective is to achieve a load balanced system, hence minimizing server idle time and maximizing system throughput. Each server has the ability to cache and discard documents based on its local load, its neighbors' loads, and on document popularity. Each server maintains an estimate of the load at its neighbors. Periodically, nodes broadcast their load to neighboring servers. If a server notices that it is overloaded, with respect to any of its neighbors, it relegates a fraction of its future predicted work to its less loaded neighbors. Specifically, the change of load at server i is determined by:

$$L_i \leftarrow L_i + \sum_{j \in N_i} \alpha_{ij} \cdot (L_{ij} - L_i),$$

where $L_i = L_{ii}$ is the current load at i , and L_{ij} is the load at j as of the last time at which j gossiped its load to i . N_i is the set of nodes in the neighborhood of i , and α_{ij} is the diffusion parameter which defines the fraction of the excess load to be exchanged between neighboring nodes ($\alpha_{ij} = 0$ if i and j are not neighbors).

Assuming that nodes have perfect information about each other's load (*i.e.*, $L_{ij}^{(t)} = L_{jj}^{(t)}$), that the diffusion overhead is zero, and that work can be transported freely among nodes, the load distribution $\bar{L}^{(t)}$ at time t can be expressed as: $\bar{L}^{(t)} = D \cdot \bar{L}^{(t-1)}$. The coefficients of the *diffusion matrix* D , can be derived straightforwardly from the above iteration. Cybenko [11],

showed that the synchronous diffusion method converges to the uniform load distribution under certain conditions, and that it does so exponentially fast. After every iteration of the diffusion algorithm, the Euclidean distance to uniform load shrinks by an amount proportional to $0 < \gamma < 1$,²

$$\|D^t \bar{L}^{(0)} - \bar{U}\| \leq \gamma^t \|\bar{L}^{(0)} - \bar{U}\|$$

where D is the diffusion matrix, $\bar{L}^{(0)}$ is the initial load distribution, \bar{U} is the uniform load distribution, and t represents time. Asynchronous diffusion also converges, as shown in Bertsekas and Tsitsiklis [3], when communication delay is bounded.

These results apply well when load diffusion is accomplished by migrating a process image, together with the resources it requires. However, document caching systems do not necessarily enable unfettered load migration, because requests need to find cache copies. Unlike typical load balancers, WebWave delegates future requests to its neighbors, rather than current load.

Hong *et al* [19], show that load balance on a hypercube can be achieved by averaging the load among neighboring processors. Xu and Lau [29], derive the optimal diffusion parameter α for a k -ary n -cube network. Lüling and Monien [23] implemented a diffusion based load balancer on a transputer with De Bruijn and ring networks.

In order to study the performance WebWave we first define our load balance goals and develop WebFold an off-line provably optimal algorithm. The sole purpose of WebFold is to study the convergence of the distributed protocol.

3 Tree Load Balance

In this section, we develop the model and definitions that will be used in Sections 4 and 5. We build our model so as to capture the tree structure T induced by the routing algorithm on the network. In terms of T , we state our goal for caching, describe the two algorithms for achieving this goal, and our arguments and evidence that we achieve it. In order to avoid requiring clients to lookup the locations of cache copies, either by contacting the home server, or even a distributed name service, we constrain the placement of copies, and hence the diffusion of load, to nodes in T . As a request for a document travels up the tree, from the node at which it originated, towards the root, it may encounter cache copies along the way. When the request flies by a node with a cache copy, the node handles it, if its present request rate is smaller than it should be.

² γ is the spectral radius of the diffusion matrix.

Table 1: Notation used in this paper.

Routing tree	Meaning	Folded tree
T	Set of tree nodes = $\{1, 2, \dots, n\}$. \mathcal{T} is a partitioning of T into contiguous regions, called <i>folders</i> .	\mathcal{T}
C_i	Children of node i .	C_i
E_i	Spontaneous request rate generated at node i .	E_i
L_i, L_{ij}	L_i = request rate served by node i , and L_{ij} is j 's request rate as known to i . $L_i = L_{ii}$.	
Δ_i	Request rate forwarded by node i to its parent. $\Delta_i = E_i + \sum_j \Delta_j - L_i$.	

Table 1 summarizes the notation used in the paper. We model the Internet as a forest of trees, each rooted at a different home server which is responsible for providing an authoritative permanent copy of some set of documents. For simplicity, we assume that every node is capable of storing an unlimited number of cached copies, and that it is willing to do so. We restrict the discussion in this paper to one tree T , considering it in isolation. As mentioned above, T captures the routes that are in effect at any point in time. A node i in T is the *parent* of j if i is the first cache server on the route from j to the home server (the root of T), in which case we also say that j is the child of i , and write $j \in C_i$.

As shown in Figure 1, every node i in the tree receives requests at the rate of $E_i + \sum_j \Delta_j$, of which it serves L_i , and forwards the rest, Δ_i , up the tree. In turn, i receives, from each child $j \in C_i$, requests that j forwards to it at a rate that we label Δ_j .

The objective of any load balancing algorithm must be defined in terms of some load metric. While the choice of workload metric can favor a particular aspect

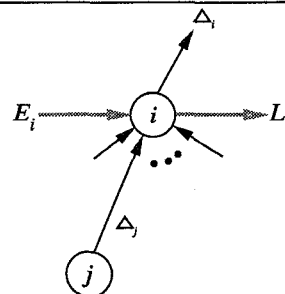


Figure 1: The flow of requests up the routing tree through a node i .

of the system being optimized [14], we choose arrival rate, because it obeys flow conservation, and thus simplifies our analysis.

On the generally true assumption that individual servers perform best when they are as lightly loaded as possible, we posit the following recursive definition of load balance.

Definition 1 (LB) A load assignment L is load-balanced iff L_{max} is minimum, and the same holds recursively if we remove the L_{max} component from L , where $L_{max} = \max L_i$.

The standard definition of *global load equality* (GLE) (i.e., $\forall i \in T, L_{max} = L_i$) follows necessarily from the above definition, when equality is indeed feasible. As we will see, load balancing over a routing tree can be sufficiently constrained to render GLE infeasible, hence the need for an alternative definition.

The first and most obvious constraint on legal load assignments over a routing tree is that the root cannot forward any load—a constraint that is not incompatible with GLE.

Constraint 1 For the root r of tree T , $\Delta_r = 0$ in any load assignment L .

The second constraint on load assignments, one that is required for there to be no need for a directory service for, or explicit discovery of, cache copies, is that of *no sibling sharing* (NSS). NSS simply states that a file can only be replicated, down the tree, in the direction of clients that request it. This effectively restricts load shifting, from a node to one of its child subtrees, to load originating in that subtree itself, and eliminates any load sharing between siblings. In other words, requests always travel upwards in the tree, towards the home server, in order to be serviced.

Constraint 2 (NSS) For every node $i \in T$, the net request rate it forwards, $\Delta_i \geq 0$.

Now we can specialize Definition 1 to routing trees.

Definition 2 (TLB) A load assignment L on a routing tree T is tree load balanced iff L is load balanced, subject to Constraints 1 and 2 above.

Figure 2 illustrates the difference between a TLB load distribution and a GLE one. Subject to Constraints 1 and 2, TLB attempts to move the load distribution as close as possible to GLE. Thus, whether a TLB load assignment yields a GLE load distribution or not, depends on the spontaneous requests originated at each server.

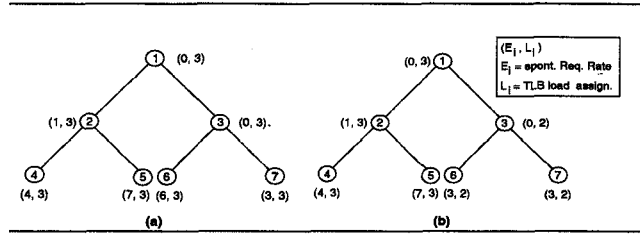


Figure 2: A tree with two different spontaneous request rates, and the corresponding TLB load assignments: (a) has a TLB load assignment that is also GLE, while (b) has a TLB load assignment that is not GLE.

4 Tree Folding

The definitions of LB and TLB above do not specify how to compute the optimal load assignment, an exercise that is necessary to evaluate load balancing protocols in practice. We approach the problem in a two-pronged manner. First we design a centralized algorithm that computes a load assignment that is provably balanced. Second, we give a diffusion-based fully distributed protocol that enables neighbors in the tree to exchange load between themselves, and give simulation evidence that the load assignment thus produced eventually converges to the optimal one. This section is devoted to the first algorithm, which we call **WebFold**, and to proving its properties—up to and including optimality.

The central insight embodied in **WebFold** is that tree nodes can be partitioned into *folds* so that every fold contains a contiguous portion of the tree that can all be assigned equal load, and no load flows between folds. Each node in a fold is allocated a load equal to the sum of all spontaneous loads in the same fold, divided by the number of nodes in the fold. That is, a node forwards load to its parent only if both are in the same fold. Folds are denoted by the names of their root nodes, and they are constructed as follows. A fold j is *foldable* (into its parent i), if the load per node in fold j exceeds that of fold i . **WebFold** repeatedly folds the foldable node with maximum per node load, until no more foldable nodes remain.

Once folding is completed, the optimal load assignment becomes obvious: spread the spontaneous load generated within a fold equally among the members of the fold. This even distribution is guaranteed not to violate NSS as specified in Constraint 2. Figure 3 shows the algorithm in complete detail.

Figure 4 demonstrates a complete folding process, solid circles represent nodes of the routing tree T and their spontaneous request rates, and dashed lines delimit the folds that make up the folded tree \mathcal{T} . Each

WebFold(\mathcal{T})

- (1) $\mathcal{T} \leftarrow T$
- (2) **foreach** $i \in T$:
 - (2.1) $\mathcal{F}_i \leftarrow \{i\}$
 - (2.2) $\mathcal{C}_i \leftarrow C_i$
 - (2.3) $\mathcal{E}_i \leftarrow E_i$
- (3) $\mathcal{T} \leftarrow \text{Fold}(\mathcal{T})$
- (4) **foreach** $j \in T$: $L_j \leftarrow \frac{\mathcal{E}_j}{|\mathcal{F}_j|}$, $j \in \mathcal{F}_i$

Fold(\mathcal{T})

- (1) **if** Empty(\mathcal{T}) **return** NIL
- (2) **while** $\exists j, i \in \mathcal{T}$, such that: Foldable(j, i), and $\frac{\mathcal{E}_j}{|\mathcal{F}_j|}$ is maximum over all foldable nodes.
 - (2.1) $\mathcal{T} \leftarrow \mathcal{T} \setminus \{j\}$
 - (2.2) $\mathcal{F}_i \leftarrow \mathcal{F}_i \cup \mathcal{F}_j$
 - (2.3) $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup \mathcal{C}_j$
 - (2.4) $\mathcal{F}_i, \mathcal{C}_i, \mathcal{E}_i \leftarrow \mathcal{F}_i \cup \mathcal{F}_j, \mathcal{C}_i \cup \mathcal{C}_j, \mathcal{E}_i + \mathcal{E}_j$
- (3) **return** \mathcal{T}

Foldable(j, i) /* Check if j is foldable into i . */

- (1) **return** [$j \in \mathcal{C}_i \wedge \frac{\mathcal{E}_j}{|\mathcal{F}_j|} \geq \frac{\mathcal{E}_i + \mathcal{E}_j}{|\mathcal{F}_j| + |\mathcal{F}_i|} \geq \frac{\mathcal{E}_i}{|\mathcal{F}_i|}$]
-

Figure 3: A provably optimal centralized algorithm. It folds together all adjacent nodes whose load can be equal.

fold's spontaneous request rate equals the sum of the spontaneous requests generated by the nodes it contains.

The following technical lemma helps in proving our central theorem. In addition, it yields insight into the operation of WebFold. Lemma 1 states that the load assignment produced by WebFold is monotonically non-increasing as we descend the routing tree T from root towards leaf.

Lemma 1 *After WebFold returns, $\forall i \in T, j \in C_i$: $L_i \geq L_j$.*

Proof: For brevity, all proofs are omitted from this paper, but can be referred to, in [18].

En route to proving that the load assignment L produced by WebFold is TLB optimal, we first prove that it satisfies Constraints 1 and 2 (NSS) respectively. Let the sequence of folds performed by Fold, and the corresponding states of the folded tree \mathcal{T} be:

$$\mathcal{T}^{(0)} \xrightarrow{f_0} \mathcal{T}^{(1)} \xrightarrow{f_1} \dots \mathcal{T}^{(n)} \xrightarrow{f_n} \mathcal{T}^{(n+1)} \longrightarrow \dots \quad (1)$$

We begin by proving that L satisfies an even stronger condition than Constraint 1, since the stricter version is useful in proving NSS. The following lemma states that WebFold distributes the load in such a way that no load is exchanged between folds.

Lemma 2 *After every fold performed by Fold, $\forall n \forall i \in \mathcal{T}^{(n)} : \Delta_i^{(n)} = 0$.*

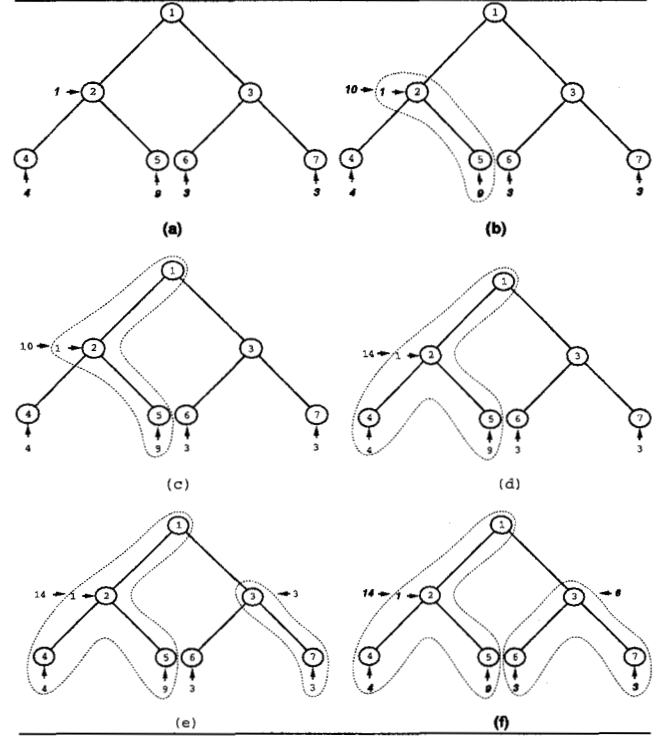


Figure 4: WebFold(\mathcal{T}) in action. A complete sequence of folds from start to finish. Note that the TLB load assignment is not GLE.

That WebFold satisfies TLB (see Definition 2), ends up hinging on the structure of the folds computed by the Fold procedure. This structure is explored in the proof of the following pivotal lemma, which establishes that WebFold chooses L in such a way as to prevent any load sharing between siblings on the routing tree (see Constraint 2).

Lemma 3 *Every load assignment L produced by WebFold satisfies the constraint of no sibling sharing (NSS).*

Theorem 1 *The load assignment L , computed by WebFold is tree load balanced.*

Now that we have established a reliable means of computing the exact load assignments that satisfy TLB, we have the tool to evaluate any load balancing distributed protocol. WebWave, which we describe in the next section, is one such protocol.

5 WebWave Protocol

In this section, the underlying protocol adopted for WebWave is presented. In addition, simulation evidence of its optimality is provided.

WebWave(T)

- (1) $\alpha_i \leftarrow \frac{1}{2+|C_i|}$ /* other values of α_i are possible */
- (2) **do forever**
 - (2.1) **foreach** j child of i in T **do**
 - /* Load can shift to/from child */
 - $L'_i \leftarrow L_i - \min\{\Delta_j, \alpha_i \cdot (L_i - L_{ij})\}$
 - (2.2) **for** k parent of i **do**
 - /* Load can shift from/to parent */
 - $L'_i \leftarrow L_i + \min\{\Delta_i, \alpha_k \cdot (L_{ik} - L_i)\}$
 - (2.3) $L_i \leftarrow L'_i$
 - (2.4) **send**(L_i) to *parent*(i) and *children*(i) in T

Figure 5: WebWave a fully distributed diffusion based algorithm.

As mentioned earlier, WebWave achieves load balancing by attempting to equalize the load between neighboring servers, subject to the *no sibling sharing* constraint. Each server i maintains L_i the number of requests serviced locally and Δ_j the number of requests it receives from child $j \in C_i$.³ In addition, i maintains L_{ik} , an estimate of the load of its neighbor $k \in (C_i \cup \text{parent}(i))$.

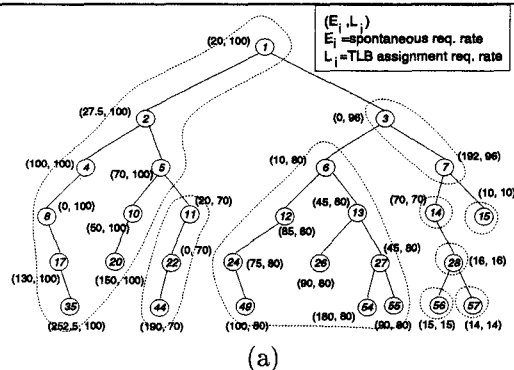
Periodically a server estimates the number of future requests that should be delegated to/from each of its children and its parent. Under NSS, the number of future requests delegated from a node i to child j cannot exceed Δ_j . Each server participating in WebWave periodically executes the algorithm in Figure 5 attempting to balance its load with its neighbors (*i.e.*, servers do not need to know the optimal load distribution). In a realistic system, WebWave servers would have two parameters: the gossip period, and the diffusion period.

When the spontaneous load pattern fortuitously allows a GLE load distribution to satisfy NSS, WebWave provably converges. This is because it satisfies the following conditions that have been shown by Cybenko [11] to be sufficient for convergence: (1) $1 - \sum_{i \in N_i} \alpha_{ij} > 0$, and, (2) the communication network is connected. However, we need also to show that WebWave converges in general to TLB; this we do by simulation.

5.1 WebWave's Convergence

We have conducted simulations to test whether WebWave converges to TLB, and how fast it does so, under a variety of conditions. All our simulation results are consistent with the proposition that WebWave is indeed optimal.

³An implementation of WebWave needs to maintain a separate Δ_j for each document it caches, which introduces a hazard that we discuss in Section 5.2.



conditions for the convergence of the diffusion method. The convergence argument relies on the fact that, on every iteration, server load assignments move monotonically closer to the uniform load. Bertsekas and Tsitsiklis [3] prove the convergence of the asynchronous diffusion method provided that communication delay is bounded. These proofs do not address tree load balance, nor do they account for per document caching effects such as potential barriers.

A number of hierarchical caching strategies were proposed in the literature, the most relevant of which are [4, 5, 9, 12, 16]. Blaze [5] and Dahlin *et al* [12] study a demand-based hierarchical caching scheme in which clients serve files from their own cache to other clients. Geographically based push-caching was proposed in [16], in this scheme servers are allowed to disseminate documents based on the geographic location of clients and on document popularity.

In [4], Bestavros analyzes two push based caching schemes, with respect to bandwidth consumption, space requirement, response time for clients, and server load sharing. Our approach differs in that we concentrate on load balancing exclusively, aiming to guarantee global optimality, instead of incremental improvements resulting from load sharing. We address protocol design questions (*e.g.*, we rule out the possibility of clients or servers consulting any cache directory), while Bestavros concentrates on evaluating resource allocation policies.

Through minor kernel modification Anderson and Patterson [1] implement user level packet filters to redirect requests to lightly loaded servers. Other approaches [21, 24] use the round-robin feature of Domain Name Servers (DNS) to distribute load among a number of Web servers. Our work differs in that it addresses a significantly larger scale of load balancing, in which it is not possible to rely on name resolution as a means of shifting load, since this introduces its own performance bottleneck.

The problem of choosing the best cache copy (or replica) to serve a particular request, is addressed by [15, 7]. Implicit in [7, 15] is the assumption that some name service can be expected to yield a list of candidate copies among which a client can choose the best. In our view, such inherently off-route global cache naming service will prove too expensive for most documents.

7 Discussion

We have set forth a formal definition of what it means for a document service to have an optimal load distribution, captured a constraint for achieving this goal in

a distributed manner, and demonstrated that a set of servers can cooperate to achieve this objective (strictly through the use of local information). Our formal definition of *tree-load balance* (TLB) expresses the optimal load distribution, for a caching system that is not allowed to lookup a cache directory, or to probe the network, to find cache copies. In order to demonstrate that TLB is achievable in a distributed fashion, first we determined the optimal load distribution using a provably optimal off-line algorithm, WebFold. Then we presented WebWave as a fully distributed diffusion based caching algorithm. WebWave implicitly determines the number and placement of cache copies as well as the number of requests allocated to each copy. Using the optimal load distribution computed by WebFold, we establish, through the use of simulation, the convergence of WebWave. Finally, we illustrate that the convergence rate of WebWave can be estimated by γ^t where $0 < \gamma < 1$.

WebWave requires routers to accept packet filtering code that extracts packets relevant to the protocol. Such injectable filters can be implemented very efficiently, as has been shown by numerous experimental efforts [1, 2, 13, 30].

Although the focus of our load balancing objective is on a single tree, it will be important, in the future, to evaluate how WebWave functions in the context of the forest of overlapping routing trees that is the Internet. Other future work includes analyzing WebWave for stability, especially under realistic load [10], and measuring its effects on network traffic and client response time.

Acknowledgements. We would like to thank Mark Crovella for his direct, constructive, criticism, and Mohammad Al-Ansari and Robert Carter for many provocative discussions. The members of the Oceans Group⁴ provided much interesting feedback when we presented our ideas in their early form. We also thank the anonymous reviewers for their valuable comments on an earlier version of this paper.

References

- [1] E. Anderson and D. Patterson. The Magicrouter: An application of fast packet interposing. Technical report, Univ. of California, Berkeley, EECS, Computer Science Division, May 1996. Available as <http://www.cs.berkeley.edu/~eanders/magicrouter/osdi96-mr-submission.ps>.
- [2] M. Bailey, B. Gopal, M. Pagels, L. Peterson, and P. Sarkar. Pathfinder: A pattern-based packet classifier. In *Proc. 1st USENIX Symp. on Op. Sys. Design and Implementation*, pages 115–123, November 1994.

⁴<http://www.cs.bu.edu/groups/oceans/>

- [3] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, Englewood Cliffs, New Jersey, 1989.
- [4] A. Bestavros. Speculative data dissemination and service to reduce server load, network traffic and service time for distributed information systems. In *Proc. 12th IEEE Intl. Conference on Data Engineering, New Orleans, Louisiana*, Mar. 1996.
- [5] M.A. Blaze. *Caching in Large-scale Distributed File Systems*. PhD thesis, Princeton Univ., Dept. of Computer Science, Jan. 1993.
- [6] J. E. Boillat. Load balancing and poisson equation in a graph. *Concurrency: Practice and Experience*, 2(4):289–313, December 1990.
- [7] R. Carter and M. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical Report BU-CS-96-007, Boston Univ., Computer Science Dept., <www.cs.bu.edu/techreports>, Mar. 1996.
- [8] John M. Chambers and Trevor J. Hastie. *Statistical Models in S*. Wadsworth & Brooks/Cole Advanced Books Software, Pacific Grove, California, 1992.
- [9] A. Chankhunthod, P.B. Danszig, C. Neerdaels, M.F. Schwartz, and K.J. Worrell. A hierarchical Internet object cache. In *Proc. USENIX Annual Technical Conference, San Diego, California*, Jan. 1996.
- [10] Mark Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. In *Proc. ACM SIGMETRICS '96*, 1996.
- [11] G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel and Distributed Computing*, 7:279–301, July 1989.
- [12] Michael D. Dahlin, Randolph Y. Wang, Thomas E. Anderson, and David A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proc. 1st USENIX Symp. on Op. Sys. Design and Implementation*, pages 267–280, November 1994.
- [13] Dawson R. Engler and M. Frans Kaashoek. DPF: Fast, flexible message demultiplexing using dynamic code generation. In *Proc. of ACM SIGCOMM'96 Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, Aug. 1996.
- [14] Domenico Ferrari and Songnian Zhou. An empirical investigation of load indices for load balancing applications. In *Proc. 12th Intl. Symp. on Computer Performance Modeling, Measurements, and Evaluation*, pages 515–528, 1987.
- [15] James Guyton and Michael Schwartz. Locating nearby copies of replicated Internet servers. Technical Report CU-CS-762-95, Department of Computer Science, University of Colorado–Boulder, February 1995.
- [16] James Gwertzman. Autonomous replication in wide-area internetworks. Technical Report TR-17-95, Harvard University, April 1995.
- [17] A. Heddaya and S. Mirdad. Wave: Wide-area virtual environment for distributing published documents. In *Electronic Proc. ACM SIGCOMM'95 Workshop on Middleware, Cambridge, Mass.*, Aug. 1995. Available as <http://www.acm.org/sigcomm/sigcomm95/workshop/>.
- [18] A. Heddaya and S. Mirdad. Webwave: Globally load balanced fully distributed caching of hot published documents. Technical Report BU-CS-96-024, Boston Univ., Computer Science Dept., <www.cs.bu.edu/techreports>, Oct. 1996.
- [19] Jiawei Hong, Xiaonan Tan, and Marina Chen. From local to global: An analysis of nearest neighbor balancing on hypercube. *Performance Evaluation Review*, 16(1):73–82, May 1988.
- [20] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Trans. on Computer Systems*, 6(1):1–32, February 88.
- [21] E.D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. In *Proc. 1st Intl. World-Wide Web Conference*, May 1994.
- [22] B. Liskov, S. Ghemawat, R. Gruber, P. Johnson, L. Shriru, and M. Williams. Replication in the Harp file system. In *Proc. 13th ACM Symp. on Operating System Principles, Asilomar, California*, pages 226–238, Oct. 1991. Also published as *Operating Systems Review*, vol. 25, no. 5.
- [23] R. Lüling and B. Monien. Load balancing for distributed branch and bound algorithms. In *Proc. 6th IEEE Intl. Parallel Processing Symp.*, pages 543–548, 1992.
- [24] J.C. Mogul. Network behavior of a busy Web server and its clients. Technical report, DEC WRL Research Report, October 1995.
- [25] D. Muntz and P. Honeyman. Multi-level caching in distributed file systems -or- your cache ain't nuthin' but trash. In *Proc. Winter USENIX Technical Conf., San Francisco, CA*, pages 305–314. USENIX, January 1992.
- [26] Michael Nelson, Brent Welch, and John Ousterhout. Caching in the Sprite network file system. *ACM Trans. on Computer Systems*, 6(1):134–154, February 1988.
- [27] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon. Design and implementation of the Sun network file system. In *Proc. Summer USENIX Technical Conf.*, June 1985.
- [28] D. Wessles and K. Claffy. Internet cache protocol (icp), version 2. Technical Report Internet-Draft, IETF Network Working Group, Nov. 1996. <[ftp://ds.internic.net/internet-drafts](http://ds.internic.net/internet-drafts)>.
- [29] C.-Z. Xu and F. C. M. Lau. Optimal parameters for load balancing using the diffusion method in k-ary n-cube networks. *Information Processing Letters*, 47(4):181–187, Sept. 1993.
- [30] M. Yahara, B. Bershada, C. Maeda, and E. Moss. Efficient packet dimultiplexing for multiple endpoints and large messages. In *Proc. Winter USENIX Technical Conf.*, 1994.