On Caching Effectiveness of Web Clusters under Persistent Connections*

Xueyan Tang† & Samuel T. Chanson
Department of Computer Science
The Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
E-mail: {tangxy, chanson}@cs.ust.hk

Abstract

Due to the emergence of the HTTP/1.1 standards, persistent connections are increasingly being used in web retrieval. This paper studies the caching performance of web clusters under persistent connections, focusing on the difference between session-grained and request-grained allocation strategies adopted by the web switch. It is shown that the content-based algorithm considerably improves caching performance over the content-blind algorithm at the request-grained level. However, most of the performance gain is offset by the allocation dependency that arises when the content-based algorithm is used at the session-grained level. The performance loss increases with cluster size and connection holding time. An optimization problem is formulated to investigate the best achievable caching performance under session-grained allocation. Based on a heuristic approach, a session-affinity aware algorithm is presented that makes use of the correlation between the requests in a session. Experimental results show that while the session-affinity aware algorithm outperforms the content-based algorithm under session-grained allocation, this optimization cannot fully compensate for the performance loss caused by allocation dependency.

1 Introduction

To handle the rapid growth of web traffic, many busy web sites are adopting cluster-based server architectures in an effort to improve the throughput rate [1, 2, 3]. A *web cluster* typically consists of a *web switch* and a collection of tightly coupled *web servers*. The web switch accepts client requests and

*The work described in this paper was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. HKUST6066/00E).

†Xueyan Tang is now with the School of Computer Engineering, Nanyang Technological University, Nanyang Technological University, Nanyang Avenue, Singapore 639798. E-mail: asxytang@ntu.edu.sg.

distributes them to the servers based on an *allocation algorithm*. Due to the speed difference between memory and disk accesses, objects stored in the server caches can be accessed much faster than those on the disks. Therefore, the performance of a web cluster, to a large extent, depends on the caching effectiveness of the web servers, which is in turn affected by the allocation algorithm of the web switch. In general, a higher cache hit ratio implies lower disk load and higher system throughput.

To achieve high caching performance, it is desirable to avoid unnecessary replication of objects in the server caches by favoring the same server for requests on the same target object [4, 5, 6]. Depending on whether web switch allocation algorithms consider request contents (i.e., the URLs of target objects) in making assignment decisions, they can be classified into *content-blind* and *content-based* algorithms [7]. Meanwhile, the control granularity of the web switch is affected by the HTTP protocol used. With the HTTP/1.0 protocol [8], the clients set up a new TCP connection for each HTTP request and the connection is closed on completion of the request. In this case, the requests can be independently assigned to the web servers. This is referred to as *request-grained* allocation. The more recent HTTP/1.1 protocol [9, 10, 11] supports the use of persistent TCP connections. Persistent connections allow a client to reuse a single TCP connection to send multiple HTTP requests. This has the advantages of amortizing the overhead of connection establishment over a group of requests and avoiding multiple TCP slow-starts [12]. We shall refer to the continuous stream of all requests sent over the same TCP connection as a *session*. With persistent connections, the allocation algorithm used by the web switch is often *session-grained* [13], where all requests on a single connection are assigned to the same server. This constraint (known as *allocation dependency*) can reduce the caching performance of web clusters. While there are some proposals on request-grained allocation under persistent connections, they involve much higher overhead than session-grained allocation (see Section 2 for details). It should be noted that the granularity of request allocation is orthogonal to whether the allocation is content-based or content-blind. Therefore, it is important to understand the relative caching performance of session-grained and request-grained strategies under both content-based and content-blind allocations in managing web clusters. Unfortunately, not much work exists in the literature on this important topic.

This paper studies the caching performance of web clusters under persistent connections. We first compare a content-based allocation algorithm with a content-blind algorithm by trace-based simulation experiments. It is shown that request-grained content-based allocation considerably improves caching performance over content-blind allocation. However, most of the performance gain is offset by the allocation dependency that arises when the content-based algorithm is used at the session-grained level. The performance loss increases with cluster size and connection holding time. We then formulate an optimization problem to investigate the best achievable caching performance under session-grained allocation granularity. Based on a heuristic approach, a session-affinity aware algorithm is proposed

that makes use of the correlation between the requests in a session. Experimental results show that the session-affinity aware algorithm outperforms the content-based algorithm under session-grained allocation considerably. However, its performance is still worse than that of request-grained content-based allocation by a significant amount. This implies optimizing session-grained allocation cannot fully compensate for the performance loss caused by allocation dependency.

The rest of this paper is organized as follows. Section 2 outlines the background of web cluster architecture. The content-based and session-affinity aware algorithms are presented in Section 3. Section 4 describes the simulation methodology, and the experimental results are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Background of Web Cluster Architecture

Many of today's popular web sites are serviced by web clusters. Figure 1 shows the high-level architecture of a web cluster, where a group of web servers are connected together to offer high request processing capacity. To keep the distributed architecture transparent to the clients, a uniform external interface of the web cluster is provided by a web switch which hides the IP addresses of the web servers. Only the IP address of the web switch is visible to the clients. On DNS lookup, the logical name of the web site (e.g., www.site.com) is translated to the IP address of the web switch. In this way, client requests are directed to the web switch which in turn distributes them to the servers using an allocation algorithm. Since the focus of our study is on caching performance of the web servers, we shall assume all contents hosted by the web cluster are accessible to every server (i.e., a request can be serviced by any server). This can be done by either replicating all contents across independent file systems running on the servers or sharing contents by means of a distributed file system [7].

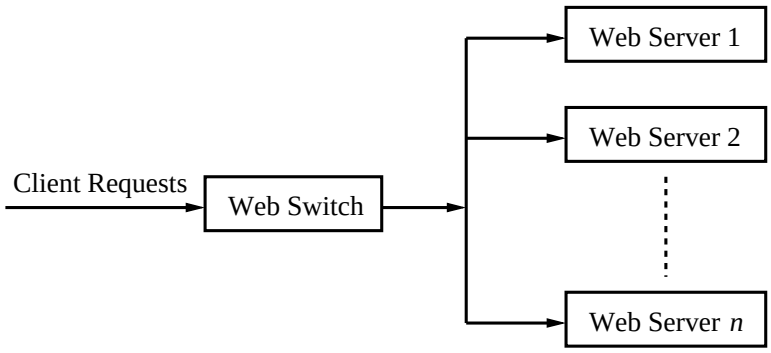


Figure 1: Web Cluster Architecture

Web switches can be classified into layer-4 switches and layer-7 switches depending on the OSI protocol layer at which they operate [3, 13]. Layer-4 switches work at the TCP (transport) level and assign new client sessions to web servers at the time of connection establishment. Upon receiving a

request packet for TCP connection setup, the layer-4 switch forwards it to a selected server and records the session-to-server mapping. The connection is then established between the client and the selected server. The web switch forwards subsequent packets in the same session to the selected server using the mapping information created. Layer-4 switches are not aware of the requested objects and cannot support content-based allocation [14]. In contrast, layer-7 switches work at the application layer. They examine the contents of HTTP requests before making allocation decisions and are therefore able to support content-based allocation. The TCP connection is initially set up between the client and the web switch. Several techniques have been proposed to allow the selected server process HTTP requests on the established connection [7]. The *TCP splicing* approach maintains persistent connections between the web switch and the web servers [15]. The web switch forwards client requests to selected servers on these connections and relays the responses from the servers to the clients. This approach does not scale up well because both inbound and outbound packets need to be processed by the web switch. The *TCP handoff* approach migrates the endpoint of the TCP connection from the web switch to the selected server [4]. In this way, outbound packets from the web servers are sent directly to the clients without passing through the web switch. The drawback is that the operating systems of both the web switch and web servers have to be modified to support TCP handoff.

Conceptually, the web switch acts as a centralized global scheduler for client requests. The allocation algorithms used by web switches can be classified into *session-grained* and *request-grained* based on its control granularity. All requests arriving on the same connection are assigned to the same server under session-grained allocation, but they can be assigned to different servers under request-grained allocation. Layer-7 switches support both session-grained and request-grained allocations, but layer-4 switches can only support session-grained allocation. Several techniques have been proposed to support request-grained allocation with layer-7 switches [16]. The *multiple handoff* approach migrates the endpoint of a connection between web servers during its lifetime. The *back-end forwarding* approach allows a web server to forward received requests to the other servers and relay the responses from these servers to the clients. Both approaches involve much higher overhead than allocating requests at the session-grained level. With non-persistent connections (i.e., the HTTP/1.0 protocol), there is no difference between session-grained and request-grained allocations due to the one-to-one correspondence between HTTP requests and TCP connections. With persistent connections (i.e., the HTTP/1.1 protocol), however, request-grained allocation is more flexible than session-grained allocation and has the potential of improving the caching performance of web servers. The objective of our study is to investigate the difference in caching performance between session-grained and request-grained allocations with and without using the content-based strategy under persistent connections.

3 Web Switch Allocation Algorithms

This section presents the allocation algorithms used in our study. For simplicity, the servers in the web cluster are assumed to have equal processing capacity.

3.1 Round-Robin Allocation

Round-Robin (RR) allocation [2, 17] is a widely used algorithm. It tries to balance the workload on web servers by assigning new requests to the servers in a circular fashion. However, while it has the advantage of simplicity, round-robin allocation does not take request contents into consideration. Thus, it is likely that requests for the same web object are sent to different servers, resulting in duplication of cache contents in the servers. For example, Figure 2 shows a stream of requests arriving at a web cluster with two servers (see the left part of the figure). The allocation result of these requests by the round-robin algorithm at request-grained level is shown in the right-hand side of Figure 2. As can be seen, for objects A, B, F and G which are accessed more than once, the requests for the same object are sent to both servers, thereby increasing the likelihood of cache misses. As will be shown in Section 5.1, the performance of RR is similar for session-grained and request-grained allocations under persistent connections. This algorithm is used as a yardstick (lower bound) on caching performance in our experiments.

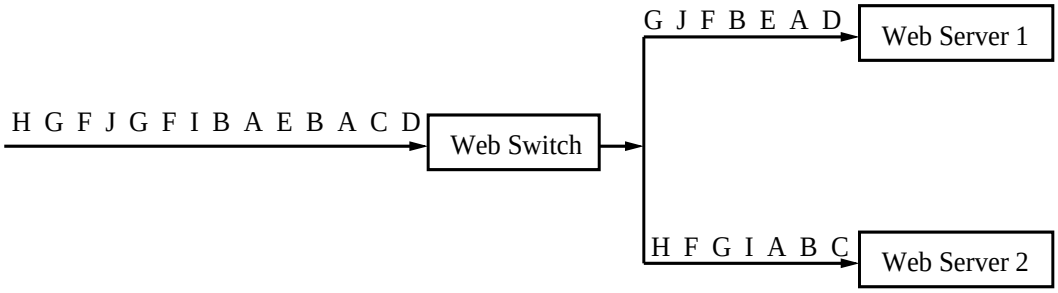


Figure 2: Example of Round-Robin Allocation

3.2 Balanced Content-Based Allocation

To improve caching performance, it is desirable to assign requests for the same object to the same server. A simple content-based allocation algorithm called *URL hashing* creates a mapping from web objects to web servers using a hash function on the URLs and assigns each request to the associated server of the target object [6]. Although URL hashing allows effective aggregation of server caches, it may suffer from unbalanced server load distribution due to skewed access pattern of web objects [18]. To allow for fair comparison with the round-robin algorithm, we present a content-based

allocation algorithm called *Balanced Content-Based (BCB) allocation* that takes server workload into consideration.

Consider request-grained allocation. Suppose there are N objects $O_1, O_2, \ldots, O_N$ served by the web cluster. Each object is associated with a load index $l(O_i)$ given by $l(O_i) = f(O_i) \cdot c(O_i)$, where $f(O_i)$ is the access frequency of O_i, and $c(O_i)$ is the load measure of serving a request for O_i. The term "load measure" is used in a general sense, which can take one of many forms such as the number of requests (in this case, $c(O_i)$ is set to 1), the number of requested bytes (in this case, $c(O_i)$ is set to the size of object O_i) and a combination of these. The problem of assigning objects to servers with the purpose of load balancing is similar to the multiprocessor scheduling problem [19]. BCB makes use of the least-load heuristic [20] to associate objects with servers. Specifically, the objects are first sorted in descending order of their load indices $l(O_i)$ and then assigned to the servers sequentially. At each step, a new object is allocated to the server with the least accumulated load. Figure 3 presents the pseudo code of the BCB algorithm, where the web cluster consists of n servers $S_1, S_2, \ldots, S_n$, and $alloc()$ is the output mapping between objects and servers. The web switch assigns each request to the designated server of the target object according to the computed mapping. Similar to URL hashing, each object is mapped to exactly one web server in the BCB allocation algorithm. Under request-grained allocation, BCB completely eliminates the duplication of cache contents in the web servers and is therefore used as an upper bound on caching performance in our experiments.

```
sort N web objects in descending order of l(O_i);
let load(S_i) = 0,   i = 1, 2, ..., n;
for j = 1 to N do {
    find server S_i with the lowest load(S_i);
    let alloc(O_j) = S_i;
    let load(S_i) = load(S_i) + l(O_j);
    }
```

Figure 3: Balanced Content-Based (BCB) Allocation

Consider again the example in Figure 2. Suppose Table 1 is the object-to-server mapping computed by the BCB algorithm, the corresponding allocation result is shown in Figure 4. It can be seen that the repetitive requests for objects A, B, F and G are each directed to the same server, thereby improving cache hit ratio at the servers.

The advantage of BCB may be reduced by session-grained allocation under persistent connections. With persistent connections, multiple HTTP requests can be submitted over a single TCP connection. Under session-grained allocation, all requests arriving on the same connection are served by the same server selected at the time of the first request arrival on that connection. This is referred to as *alloca-*

object	server	object	server
A	S_2	F	S_2
B	S_1	G	S_1
C	S_1	H	S_2
D	S_2	I	S_2
E	S_1	J	S_1

Table 1: Example Mapping from Web Objects to Web Servers

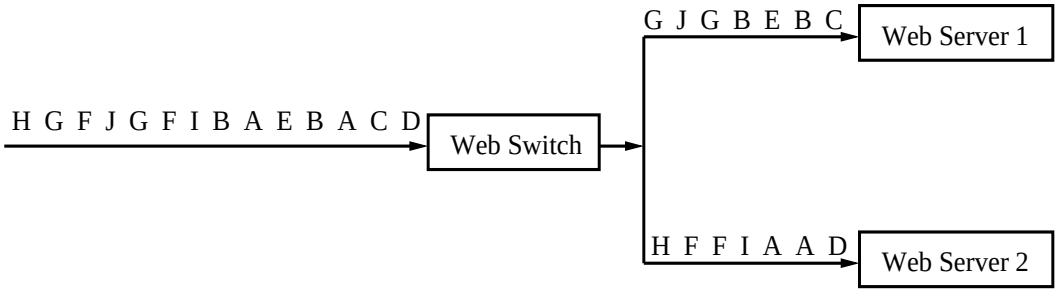


Figure 4: Example of BCB under Request-Grained Allocation

tion dependency. To facilitate presentation, we shall classify sessions into different types based on their first requests. Specifically, all sessions with the first request targeting for an object O belong to the same *session type* and are referred to as O-sessions. The BCB algorithm shown in Figure 3 can be used to handle session-grained allocation in a straightforward fashion by replacing $l(O_i)$ with $l(O_i\text{-session})$, the load index of O_i-sessions. $l(O_i\text{-session})$ is computed by taking the product of $f(O_i\text{-session})$ and $c(O_i\text{-session})$, where $f(O_i\text{-session})$ is the arrival rate of O_i-sessions, and $c(O_i\text{-session})$ is the load measure of serving an O_i-session.[1] Accordingly, the mapping computed by the BCB algorithm associates session types with web servers. The web switch assigns each new session to the designated server based on its session type. Consider the example in Figure 4. Assume that the incoming requests to the web cluster arrive in four different connections (see the dashed boxes in the left-hand side of Figure 5). Taking the hypothetical mapping of Table 1 and replacing each object O by O-session, the corresponding allocation result under session-grained allocation is shown in the right-hand side of Figure 5. As can be seen, the advantage of content-based allocation is significantly weakened. The requests for the same object (A, B, F and G) are distributed to both servers as in the round-robin algorithm. This can result in unnecessary replication of cache contents in the servers.

[1]For example, if the load measure takes the form of the number of requests, $c(O_i\text{-session})$ is set to the average number of requests in an O_i-session. If the load measure is in the form of the number of requested bytes, $c(O_i\text{-session})$ is set to the mean aggregate size of all objects requested in an O_i-session.

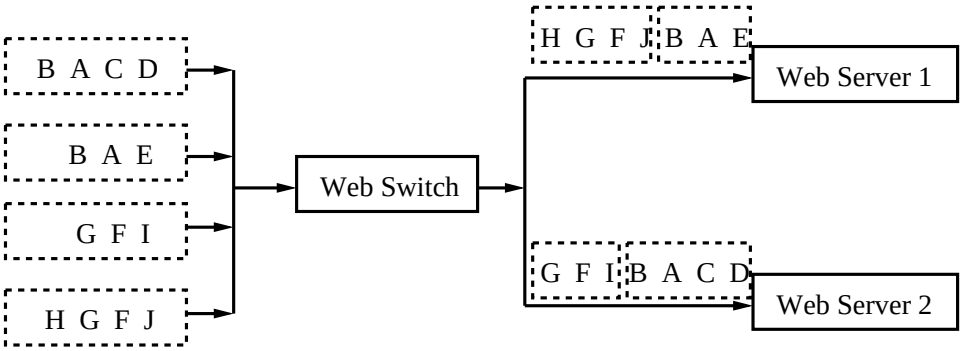


Figure 5: Example of BCB under Session-Grained Allocation

3.3 Session-Affinity Aware Allocation

The BCB algorithm does not produce the best caching performance under session-grained allocation. Consider again the example shown in Figure 5. If the D-session and the E-session are allocated to the first server and the I-session and the J-session are allocated to the second server (see Figure 6), the repetitive requests for objects A, B, F and G would be sent to the same server. This shows the importance of considering session "similarity" in session-grained allocation, where the "similarity" between sessions is measured by the degree of overlap in the requested contents. Two sessions have higher "similarity" if they have more requested contents in common. For example, the D-session and E-session in Figure 6 have high "similarity" because objects A and B are accessed in both sessions (i.e., two common requests out of four requests). Similarly, the I-session and J-session also have high "similarity" because they have two requests for F and G in common out of four requests. By allocating "similar" sessions to the same server, the caching effectiveness of the web cluster can be improved. This is an extension of the concept of content-based allocation at the request-grained level.

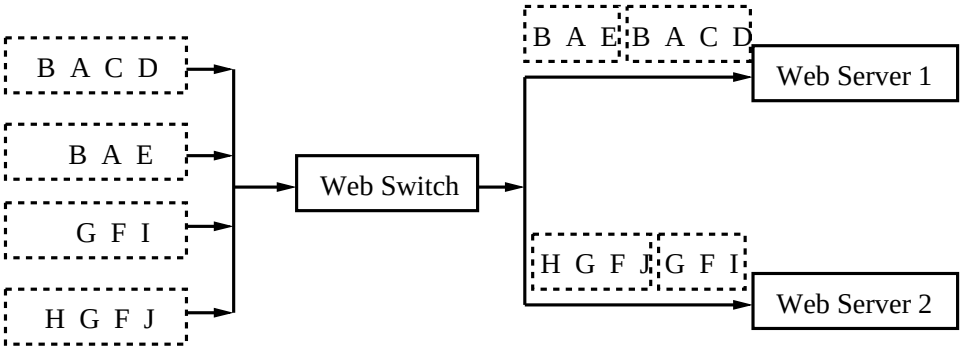


Figure 6: Improved Session-Grained Allocation under Persistent Connections

Recall that under session-grained allocation, all requests arriving on the same connection are assigned to the same server selected on the first request arrival of the session. Therefore, only the first request in a session is known at the time of session assignment. Thus, an important issue of considering

session "similarity" is whether session "similarity" can be predicted, to some extent, based on the first requests of sessions. We have examined the correlation between the first and subsequent requests in a session using web server traces (see Section 4 for details of these traces). For each session-object pair (O_i-session, O_j), the *correlation coefficient* $r(O_j \mid O_i\text{-session})$ is defined as the relative access frequency of object O_j in O_i-sessions, i.e.,

$$r(O_j \mid O_i\text{-session}) = \frac{f(O_j \mid O_i\text{-session})}{f(O_i\text{-session})},$$

where $f(O_j \mid O_i\text{-session})$ is the arrival rate of requests for O_j in O_i-sessions and $f(O_i\text{-session})$ is the arrival rate of O_i-sessions. It is obvious that the larger the coefficient, the higher the correlation between O_i-session and object O_j. For each web server trace, we computed the correlation coefficients of all session-object pairs (O_i-session, O_j) such that $O_i \neq O_j$ and the O_i-session is among the most popular 20% session types.[2] The coefficients equal to 0 were then discarded and the remaining coefficients were sorted in descending order. As shown in Figure 7, the correlation coefficients exhibit a heavy-tailed property. Some session-object pairs have very high correlation coefficients. For example, the top 600 pairs have correlation coefficients above 0.6. Meanwhile, there are also a high proportion of pairs with correlation coefficients close to 0. This behavior can be explained as follows. On the one hand, clients retrieving an HTML page would normally request the embedded images shortly afterwards. Therefore, sessions whose first request is for an HTML page P are expected to have high correlation coefficients with the embedded images of P. On the other hand, human browsing behaviors such as hyperlink selection are less predictable. Sessions starting with an HTML page P may not have high correlation coefficients with the pages pointed at by P. The results shown in Figure 7 imply there is some correlation between the first request and the subsequent requests in a session. A session-grained allocation algorithm can exploit this correlation to improve caching performance.

In the following, we formulate an optimization problem to investigate the best achievable caching performance of session-grained allocation. We take the byte hit ratio (i.e., the number of bytes served from the caches relative to the total requested bytes) as a performance metric in our analysis. Note that the byte hit ratio reflects the amount of reduction in disk access. Let c_j be the cache size of server S_j, and I_j be the set of session types allocated to server S_j. It is easy to see that the expected access frequency of an object O at server S_j is given by

$$f(O \mid I_j) = \sum_{O_k\text{-session} \in I_j} f(O \mid O_k\text{-session}).$$

Let $O_{j_1}, O_{j_2}, \ldots, O_{j_N}$ be the list of all objects sorted in descending order of $f(O \mid I_j)$. To enhance caching performance, the most frequently accessed objects are normally kept in the cache. Therefore,

[2]The arrival rates of session types are highly skewed. In order not to bias the results by infrequent session types, only the most popular 20% session types are considered. They account for over 90% of all sessions in the traces we studied.

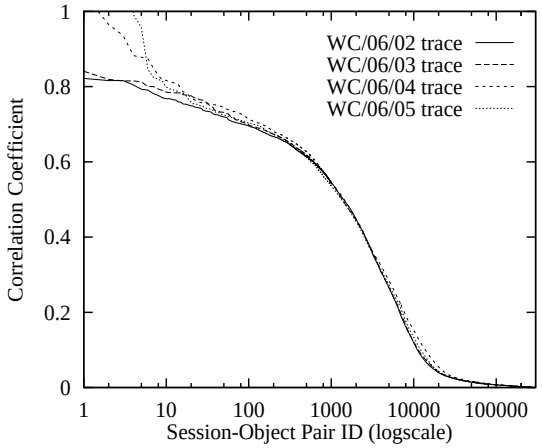


Figure 7: Correlation Coefficients

the expected byte hit ratio of server S_j is given by the following function of I_j and c_j:

$$bhr(I_j, c_j) = \frac{\sum_{i=1}^{m} f(O_{j_i} \mid I_j) \cdot s(O_{j_i}) + f(O_{j_{(m+1)}} \mid I_j) \cdot (c_j - \sum_{i=1}^{m} s(O_{j_i}))}{\sum_{i=1}^{N} f(O_i) \cdot s(O_i)}, \tag{1}$$

where $s(O)$ is the size of object O, N is the total number of objects, $f(O) = \sum_{k=1}^{N} f(O \mid O_k\text{-session})$ is the total access frequency of object O, $\sum_{i=1}^{m} s(O_{j_i}) \leq c_j$ and $\sum_{i=1}^{m+1} s(O_{j_i}) > c_j$. The numerator in equation (1) essentially computes the optimal solution of a fractional knapsack problem, and is normalized by the total requested bytes made to the web cluster. Suppose there are n web servers in the cluster. The byte hit ratio of the web cluster is given by:

$$\sum_{j=1}^{n} bhr(I_j, c_j).$$

Given c_j $(j = 1, 2, \ldots, n)$, $f(O_i)$, $s(O_i)$ and $f(O_i \mid O_k\text{-session})$ $(i, k = 1, 2, \ldots, N)$, the objective of the optimal session-grained allocation problem is to partition the N session types O_1-session, O_2-session, $\ldots$, O_N-session into n sets I_1, I_2, $\ldots$, I_n such that the overall caching effectiveness

$$\mathcal{E}(I_1, I_2, \ldots, I_n) = \sum_{j=1}^{n} bhr(I_j, c_j)$$

is maximized. This optimization problem is computationally expensive. Since each session type has n allocation choices, there are a total of n^N possible partitions. The search space is huge even for small values of n and N. Therefore, we propose a simple greedy heuristic to find a suboptimal mapping between session types and web servers. At each step, a new session type is assigned to the server that maximizes the gain in overall caching effectiveness. For the purpose of load balancing, only those servers whose accumulated load is lower than the average load of all servers (i.e., $\sum_{i=1}^{N} l(O_i\text{-session})/n$) need to be considered. The algorithm, referred to as *Session-Affinity Aware (SAA) allocation*, is given in Figure 8.

let $load(S_i) = 0, \quad i = 1, 2, \ldots, n$;
let $I_i = \phi, \quad i = 1, 2, \ldots, n$;
let $avgload = \sum_{i=1}^{N} l(O_i\text{-session})/n$;
for $i = 1$ to N do {
    find server S_j that maximizes the gain in caching effectiveness
        $\mathcal{E}(I_1, \ldots, I_j \cup \{O_i\text{-session}\}, \ldots, I_n) - \mathcal{E}(I_1, \ldots, I_j, \ldots, I_n)$,
        and $load(S_j) < avgload$;
    let $alloc(O_i\text{-session}) = S_j$;
    let $load(S_j) = load(S_j) + l(O_i\text{-session})$;
    let $I_j = I_j \cup \{O_i\text{-session}\}$;
    }

Figure 8: Session-Affinity Aware (SAA) Allocation

Note that the increase in caching effectiveness is given by

$$\begin{aligned} & \mathcal{E}(I_1, \ldots, I_j \cup \{O_i\text{-session}\}, \ldots, I_n) - \mathcal{E}(I_1, \ldots, I_j, \ldots, I_n) \\ = \quad & bhr(I_j \cup \{O_i\text{-session}\}, c_j) - bhr(I_j, c_j). \end{aligned}$$

The computation of $f(O_k \mid I_j \cup \{O_i\text{-session}\})$ $(k = 1, 2, \ldots, N)$ given $f(O_k \mid I_j)$ has a time complexity of $O(N)$. Moreover, the worst case time complexity of reordering the objects in decreasing $f(O_k \mid I_j \cup \{O_i\text{-session}\})$ based on the existing order in decreasing $f(O_k \mid I_j)$ is given by $O(N \log N)$. Therefore, the worst case computational complexity of $bhr(I_j \cup \{O_i\text{-session}\}, c_j) - bhr(I_j, c_j)$ is given by $O(N \log N)$. Thus, the allocation of each session type has a time complexity of $O(nN \log N)$, and the total time complexity of the SAA algorithm is $O(nN^2 \log N)$. Note that this is a worst case result. In most of our simulation experiments described in Sections 4 and 5, the SAA algorithm took only a few minutes to compute the mapping between session types and servers.

4 Simulation Methodology

The trace-based simulation technique was used in our experiments. This section outlines the simulation model, the characteristics of the traces, and the cache replacement algorithms used in the experiments.

We made use of the WorldCup98 web server logs [21] collected between 1998-6-2 and 1998-6-5 (four traces, one per day) to simulate the requests made to the web cluster. Each trace entry includes the request time, the client ID,[3] the target URL, the size of the target object and other information about the requests. Table 2 shows some characteristics of these traces. Interested readers are referred to [22] for more details of the WorldCup98 web server logs.

[3]It may be a proxy.

Trace	Number of Requests	Number of Distinct Objects Accessed	Number of Clients
WC/06/02	7188041	4923	72168
WC/06/03	7894566	5035	77725
WC/06/04	8297253	5939	74203
WC/06/05	8093068	5822	76197

Table 2: Trace Characteristics

The publicly available server traces are dominated by HTTP/1.0 traffic.[4] Therefore, to construct client sessions under persistent connections, we used the following simple heuristic similar to that employed in [16]: A request arriving within a fixed interval (called the *connection holding time*) of the precedent request from the same client is assigned the same connection used by the precedent request. Otherwise, the request initiates a new connection. The default value of the connection holding time was set at 15 seconds, a typical value used by web servers to close idle connections [23]. The performance impact of connection holding time is investigated in Section 5.4. In the default experimental setup, each client establishes only one persistent connection to the web cluster at a time. The case of concurrent multiple connections is studied in Section 5.5.

As shown in Figure 1, we simulated a web cluster consisting of a web switch and n web servers (n is known as the *cluster size*). The default cluster size was set at 4. The performance impact of cluster size is discussed in Section 5.3. Following common practice, the cache size of each web server is specified relative to the total size of all distinct contents accessed [24, 25]. A wide range of cache size was examined in our experiments (see Section 5.2).

Two different replacement algorithms have been implemented: a simple LRU algorithm and a generalized LFU algorithm. The generalized LFU algorithm (GLFU) is similar to the one proposed in [24]. For each object O, we refer to the product $f(O) \cdot s(O)$ as the *generalized size* (GSize) of the object, where $f(O)$ and $s(O)$ are the access frequency of O at the local server and the size of O respectively. The GSize value reflects the benefit of caching the object, i.e., the reduction in disk access. To accommodate a new object, one or more objects may need to be removed from the cache. In order to maximize the overall caching benefit, the replacement candidates should be selected such that they involve the least total benefit while creating enough space. This is equivalent to the knapsack problem which is NP-complete [19]. A fast greedy heuristic for cache replacement is as follows. The objects in the cache are arranged in increasing order of access frequency. When an object P not in the cache is accessed, the cached objects are examined and marked sequentially until sufficient space for P is created. The total GSize of the marked objects is then compared with the GSize of P. If the

[4] More than 80% of the requests in our traces used HTTP/1.0.

GSize of P is larger, the marked objects are removed and P is inserted into the cache. Otherwise, the cache contents are not updated since the benefit of caching P is outweighed by the performance loss of removing the marked objects. The access frequency of an object is dynamically estimated using a "sliding window" technique [26] covering the K most recent references of the object, i.e., $f(O) = \frac{K}{t-t_K}$, where t is the current time and t_K is the Kth most recently referenced time. K was set at 3 in our experiments. For the purpose of GSize evaluation, the reference times of objects not in the cache must be maintained. They are stored in the form of object descriptors in an auxiliary cache whose size is negligible compared to the main cache that stores the actual objects. Simple LFU replacement algorithm is used to manage the object descriptors. In our simulation experiments, the size of the auxiliary cache (in terms of the number of object descriptors) was set at 20% of the number of all distinct objects accessed.

The BCB and SAA allocation algorithms require the computation of load indices. For the experiments reported in this paper, the number of requested bytes was taken as the load indicator. Our experimental results with other load measures (e.g., number of client sessions, number of requests, and a combination of these measures) showed similar performance trends in caching effectiveness and are therefore not shown due to space limitation.

Each simulation run was performed using two daily traces $\mathcal{A}$ and $\mathcal{B}$. The RR allocation algorithm is directly evaluated using trace $\mathcal{B}$. The BCB and SAA allocation algorithms make use of the load measure and session affinity statistics collected from trace $\mathcal{A}$ to compute the allocation mapping which is then evaluated using trace $\mathcal{B}$. All caches were emptied at the start of each simulation run. The first half of trace $\mathcal{B}$ was used to drive the system into a steady state. Performance statistics were collected during the second half of trace $\mathcal{B}$ only. Two sets of experiments were performed in our simulation. In the first set, we used the same trace for both $\mathcal{A}$ and $\mathcal{B}$, representing the case where the allocation mapping is computed with *a priori* knowledge of session affinity. In the second set, two consecutive daily traces were used for $\mathcal{A}$ and $\mathcal{B}$ respectively, representing the case where the allocation mapping is recomputed on a daily basis. In this case, the session types appearing in trace $\mathcal{B}$ but not in trace $\mathcal{A}$ (which made up less than 0.1% of all sessions in our experiments) were randomly assigned to the servers. Both sets of experiments showed similar performance trends. In the next section, we shall only report the experimental results where $\mathcal{A}$ and $\mathcal{B}$ were assigned traces WC/06/02 and WC/06/03 respectively.

5 Experimental Results

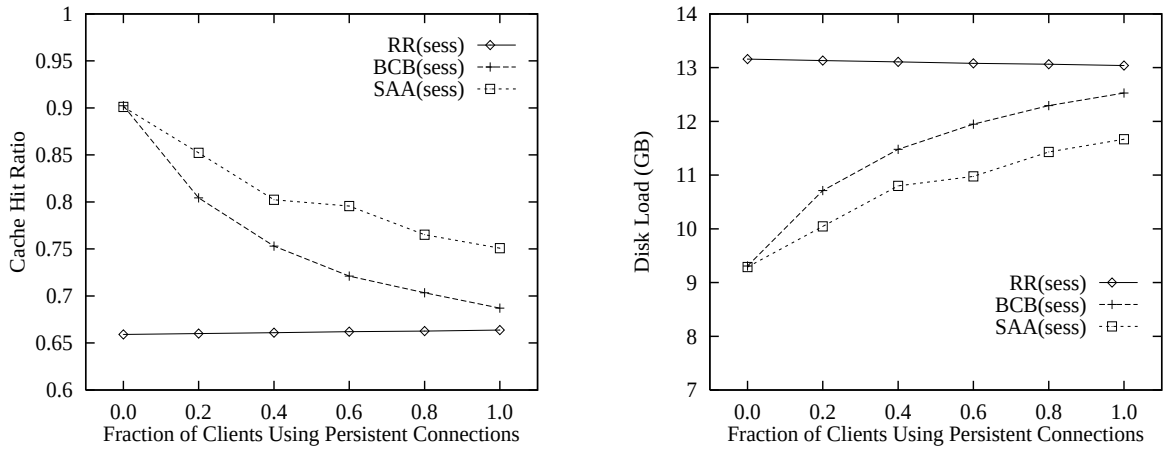
The performance metrics used in our experimental study are the cache hit ratio and disk load. The cache hit ratio measures the proportion of client requests served from the memory caches of web

servers. The disk load measures the total number of requested bytes served from the disks (i.e., those not found in the caches) and is proportional to one minus the byte hit ratio.

The allocation granularity is specified in parentheses following the algorithm. For example, BCB(sess) and BCB(req) represent the BCB allocation algorithm at the session-grained and request-grained levels respectively.

5.1 Impact of Persistent Connections

First, we examine the impact of persistent connection on session-grained allocation by varying the proportion of clients using persistent connections from 0.0 to 1.0. Figures 9 and 10 show the experimental results under LRU and GLFU replacement policies respectively. In this set of experiments, the cluster size was set at 4 and the cache size of each server was 1%. Note that the left-most points in Figures 9 and 10 (i.e., the case of non-persistent connections) represent the performance of request-grained allocations: RR(req), BCB(req) and SAA(req).

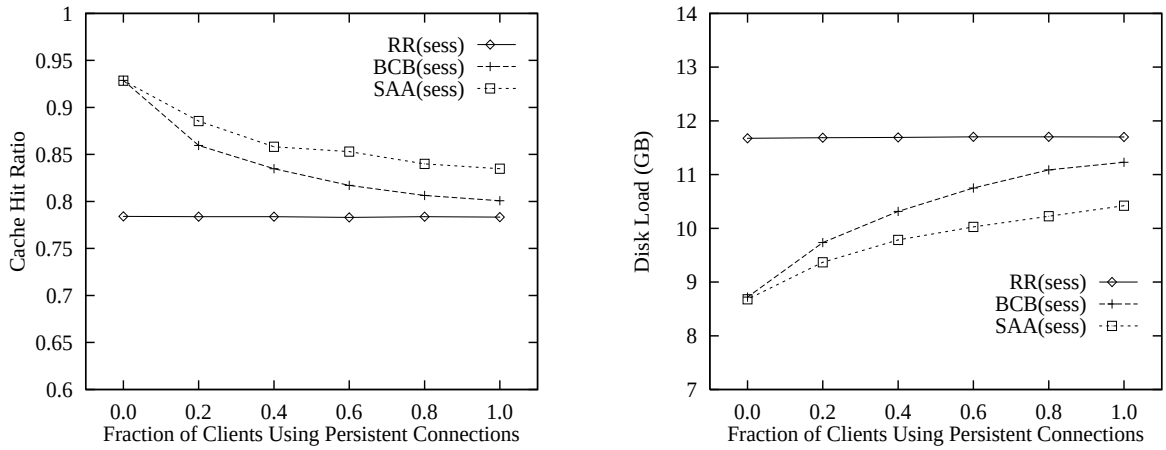


(a) Cache Hit Ratio

(b) Disk Load

Figure 9: Impact of Persistent Connections under LRU replacement

It can be seen from Figure 9(a) that the hit ratios of BCB(sess) and SAA(sess) are similar under non-persistent connections. They are much higher than that of RR(sess). Therefore, in this case, the disk loads of BCB(sess) and SAA(sess) are considerably lower than RR(sess) (see Figure 9(b)). However, the caching performance of the content-based algorithm BCB(sess) decreases rapidly with increasing use of persistent connections. As shown in Figure 9, under the LRU replacement policy, the hit ratio of BCB(sess) dropped by 24% and the disk load of BCB(sess) increased by 35% when all clients used persistent connections. This verifies the negative impact of allocation dependency on caching effectiveness under session-grained allocation. In contrast, persistent connections have little impact on the performance of content-blind algorithms such as RR(sess). On the other hand, by taking into consideration the probabilities of accessing different objects in the session, the session-



(a) Cache Hit Ratio

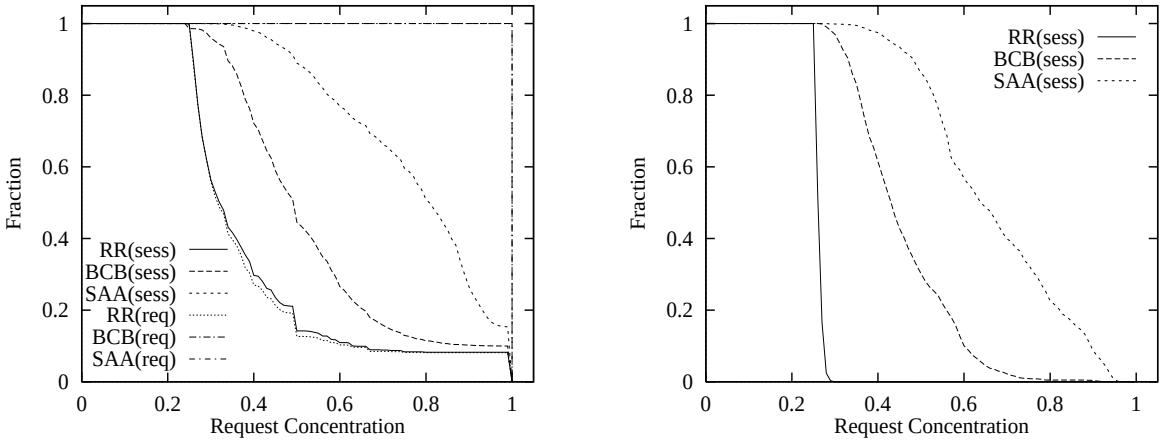
(b) Disk Load

Figure 10: Impact of Persistent Connections under GLFU replacement

affinity aware algorithm SAA(sess) outperforms BCB(sess) when some clients start to use persistent connections. The performance trends for LRU and GLFU replacement policies are similar, except that the absolute hit ratios of GLFU are higher than those of LRU.

The performance of an allocation algorithm depends on the extent to which requests for the same object concentrate on the same server — the higher the concentration, the better the caching performance. This is shown in Figure 11. For each object, we computed the highest proportion of requests received by a single server as a measure of request concentration for that object. For instance, suppose the fractions of requests for object O received by the four servers are 0.2, 0.5, 0.1 and 0.2 respectively, then the request concentration of object O is 0.5. We analyze the following six cases: RR(sess), BCB(sess) and SAA(sess) when all clients use persistent connections, and RR(req), BCB(req) and SAA(req). Figure 11(a) shows the distribution of request concentration for all objects. A point (x, y) on the curve means that a fraction y of all objects have request concentration higher than x. An allocation algorithm is expected to achieve higher caching performance if its curve is closer to the upper-right corner. It can be seen from Figure 11(a) that the graphs of BCB(req) and SAA(req) overlap with each other completely. They both contain straight line segments from $(0, 1)$ to $(1, 1)$ and then to $(1, 0)$. This is because in BCB(req) and SAA(req), all requests for the same object are sent to the same server, resulting in a request concentration of 1 for all objects. In contrast, the request concentration of RR(req) is much lower than those of BCB(req) and SAA(req). Therefore, as shown by the left-most points in Figures 9 and 10, BCB(req) and SAA(req) significantly outperform RR(req). Furthermore, the similarity between the RR(req) and RR(sess) curves in Figure 11(a) explains why the caching performance of RR(sess) is hardly affected by the use of persistent connections. Comparing the three curves for session-grained allocation, it is clear that SAA(sess) has the highest request concentration while RR(sess) has the lowest concentration. Figure 11(b) shows the distribution of

request concentration for the most frequently accessed 1000 objects. These objects are more likely to be cached and therefore have greater impact on caching effectiveness than the other objects. As can be seen, the differences in request concentration among RR(sess), BCB(sess) and SAA(sess) are even higher for the popular objects. For example, the proportions of objects having request concentration higher than 0.5 are 86%, 30% and 0% for SAA(sess), BCB(sess) and RR(sess) respectively. This explains why SAA(sess) outperforms BCB(sess) and RR(sess) under persistent connections (see Figures 9 and 10).



(a) for all objects

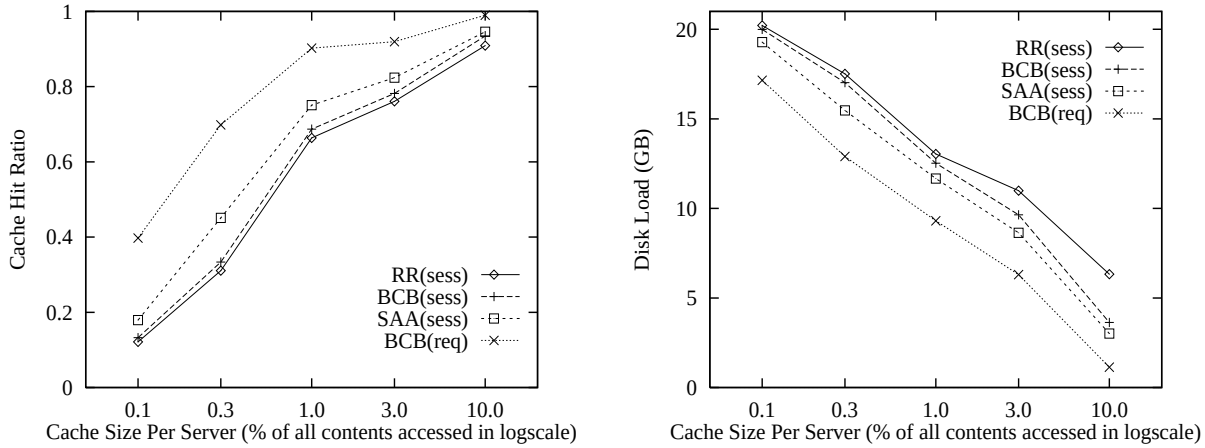
(b) for the most popular 1000 objects

Figure 11: Distribution of Request Concentration

5.2 Impact of Cache Size

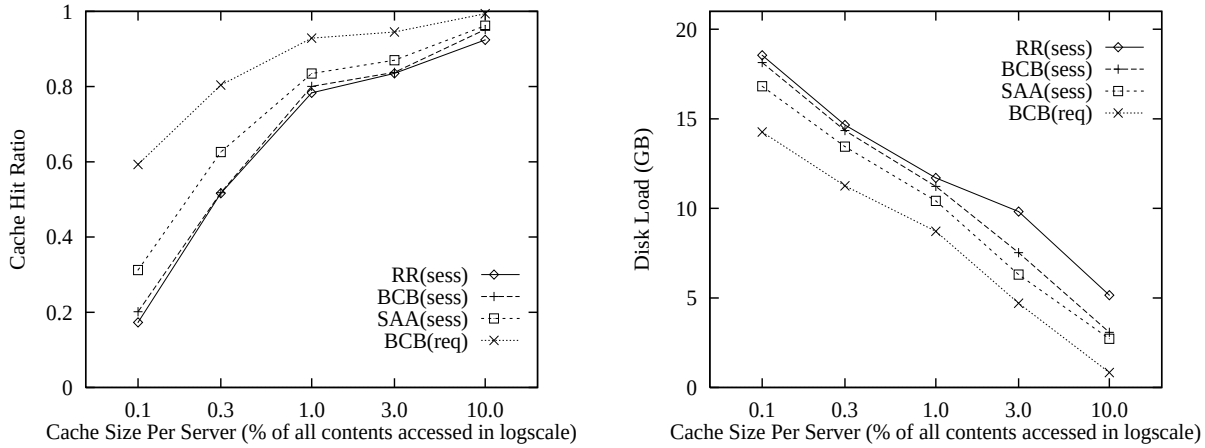
Figures 12 and 13 show the cache hit ratios and disk loads for different cache sizes. As seen in Section 5.1, the caching performance of RR(req) does not differ significantly from that of RR(sess) under persistent connections. Furthermore, the performance of BCB(req) and SAA(req) is similar. Therefore, we shall only report the results of RR(sess), BCB(sess) and SAA(sess) under persistent connections as well as BCB(req) in the rest of this paper. Note that RR(sess) and BCB(req) are used as yardsticks (lower and upper bounds respectively) on caching performance.

It is seen by comparing the graphs of BCB(req) and BCB(sess) in Figures 12 and 13 that session-grained allocation significantly reduces the caching effectiveness of BCB compared to request-grained allocation. To achieve the same hit ratio, BCB(sess) would need more than 3 times the cache space of BCB(req) (note that the cache size axis is in logscale). In fact, the hit ratios of BCB(sess) are fairly close to those of the content-blind algorithm RR(sess), especially for small cache sizes. This implies session-grained allocation offsets most of the performance benefits of content-based algorithms. As shown in Figure 12(b), BCB(sess) results in 1.5 and 3.2 times the disk load of BCB(req) for cache sizes of 3% and 10% per server respectively. Similar performance trends are also observed for the



(a) Cache Hit Ratio (b) Disk Load

Figure 12: Impact of Cache Size under LRU replacement



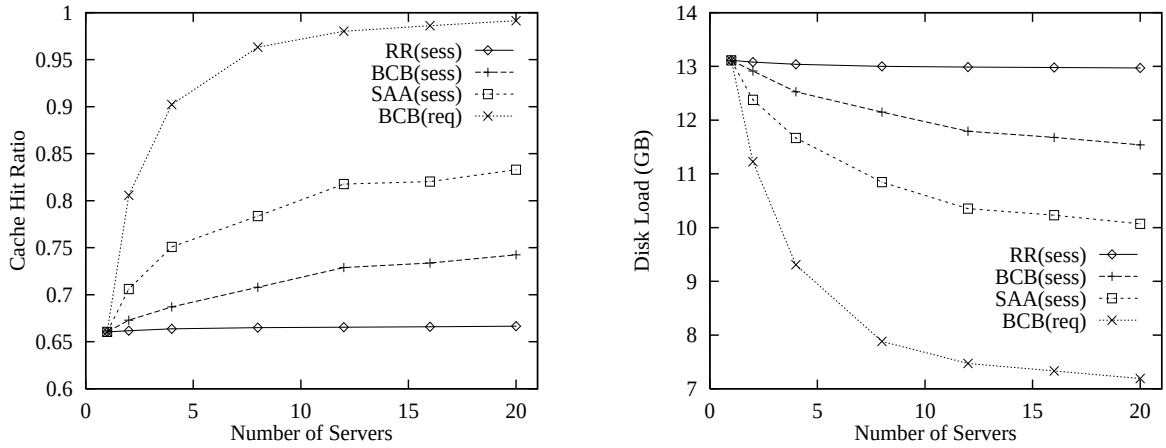
(a) Cache Hit Ratio (b) Disk Load

Figure 13: Impact of Cache Size under GLFU replacement

GLFU replacement policy (see Figure 13(b)). On the other hand, the session-affinity aware algorithm SAA(sess) maintains considerable performance gain compared to RR(sess) over a wide range of cache sizes. Since the experiments were performed with two consecutive daily traces (see Section 4), the results imply that the correlation between the first and subsequent requests within a session is relatively stable across days. Thus, historical correlation statistics can be used to optimize the performance of session-grained allocation for web clusters. SAA(sess) partially compensates for the loss in caching effectiveness under persistent connections. However, as shown in Figures 12 and 13, there remains a significant performance gap between SAA(sess) and BCB(req).

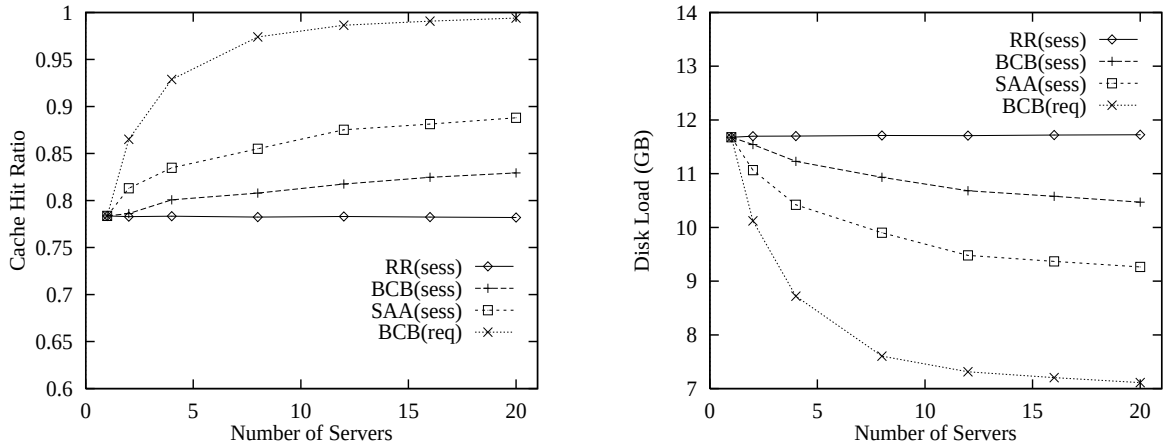
5.3 Impact of Cluster Size

To investigate the impact of cluster size, we varied the number of web servers from 1 to 20. Figures 14 and 15 show the performance results for different cluster sizes. In these experiments, the cache size of each server was set at 1%.



(a) Cache Hit Ratio (b) Disk Load

Figure 14: Impact of Cluster Size under LRU replacement



(a) Cache Hit Ratio (b) Disk Load

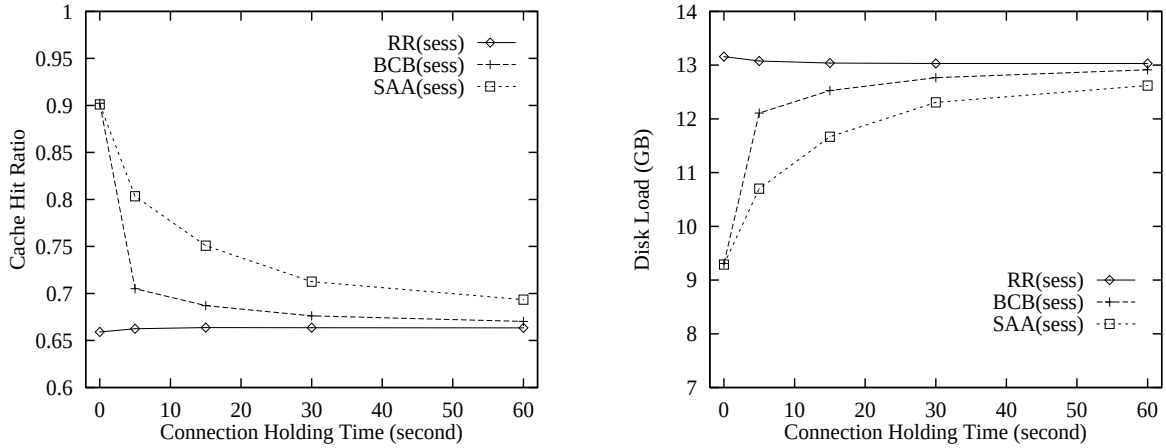
Figure 15: Impact of Cluster Size under GLFU replacement

As can be seen, the allocation algorithms have the same performance when there is only one server in the cluster. Since RR(sess) sends each request to any server in the cluster with equal probability, the caching effectiveness of web cluster is essentially the same as that of a single server, independent of the cluster size. On the other hand, since each object has at most one copy stored in the caches in BCB(req), the caches of web servers are effectively aggregated. Therefore, the hit ratio of BCB(req) improves rapidly with increasing number of servers. Due to the allocation dependency,

the hit ratio of BCB(sess) improves much slower compared to BCB(req) as the cluster size grows. The performance difference between BCB(req) and BCB(sess) generally increases with cluster size. This further demonstrates the negative impact of allocation dependency on caching effectiveness. As seen from Figures 14 and 15, SAA(sess) outperforms BCB(sess) when there are multiple servers in the cluster. The improvement of SAA(sess) over RR(sess) is maintained at about 50% of that of BCB(req) over RR(sess) (i.e., the upper bound).

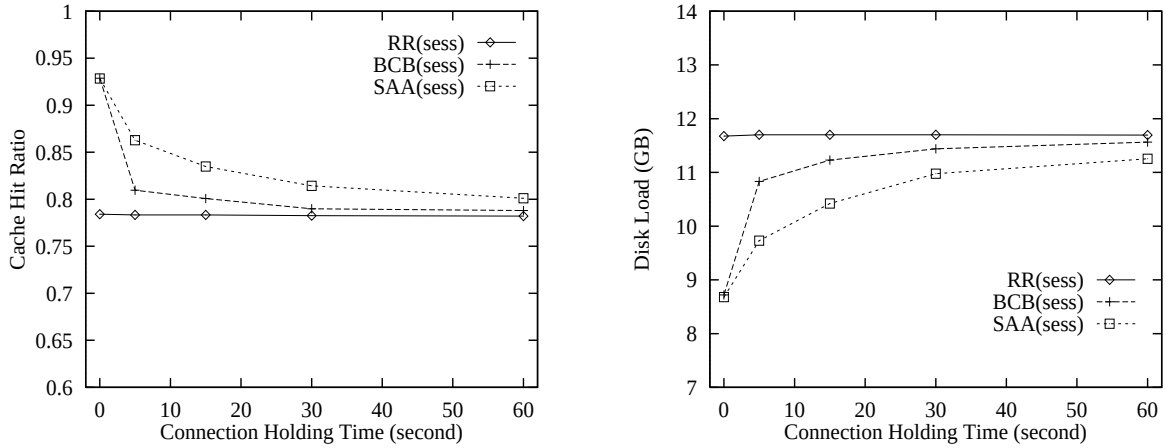
5.4 Impact of Connection Holding Time

So far, the allocation algorithms have been evaluated using the default connection holding time of 15 seconds. Figures 16 and 17 show the impact of connection holding time on cache hit ratio and disk load. Note that a holding time of 0 second represents the case of non-persistent connections.



(a) Cache Hit Ratio (b) Disk Load

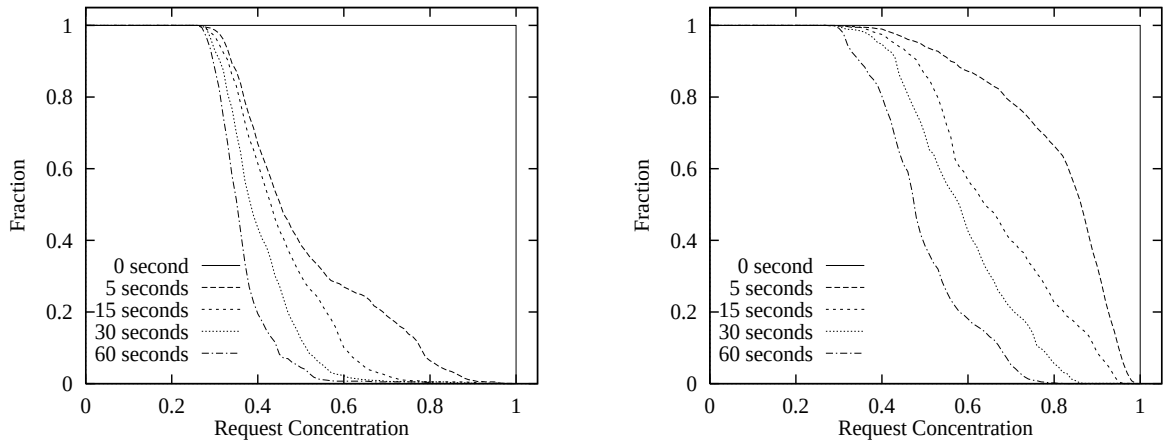
Figure 16: Impact of Connection Holding Time under LRU replacement



(a) Cache Hit Ratio (b) Disk Load

Figure 17: Impact of Connection Holding Time under GLFU replacement

It is intuitive that the average number of requests in a session increases with connection holding time, and the larger the number of requests in a session, the higher the allocation dependency. Therefore, as shown in Figures 16 and 17, the caching effectiveness of BCB(sess) decreases with increasing connection holding time. Note that even a small increase in holding time would reduce the hit ratio and push up the disk load of BCB(sess) substantially when the holding time is short (e.g., less than 15 seconds). Compared to the case of non-persistent connections, BCB(sess) shows a 22% drop in hit ratio and a 30% increase in disk load with a holding time of 5 seconds under the LRU replacement policy. Figure 18 shows the impact of connection holding time on request concentration for the most popular 1000 objects. It can be seen that the request concentration of BCB(sess) drops rapidly as the holding time increases. The proportions of request concentration higher than 0.5 are 39%, 30%, 12% and 5% for holding times of 5, 15, 30 and 60 seconds respectively. In contrast, the request concentration of SAA(sess) decreases at a much lower rate than BCB(sess). The corresponding proportions of request concentration higher than 0.5 are 94%, 86%, 70% and 39% for SAA(sess). Thus, SAA(sess) achieves higher caching performance than BCB(sess) (see Figures 16 and 17). However, the decreasing performance of SAA(sess) with increasing holding time implies optimizing session-grained allocation cannot fully compensate for the performance loss caused by allocation dependency. Figures 16 and 17 also show that connection holding time has little impact on the content-blind algorithm RR(sess).



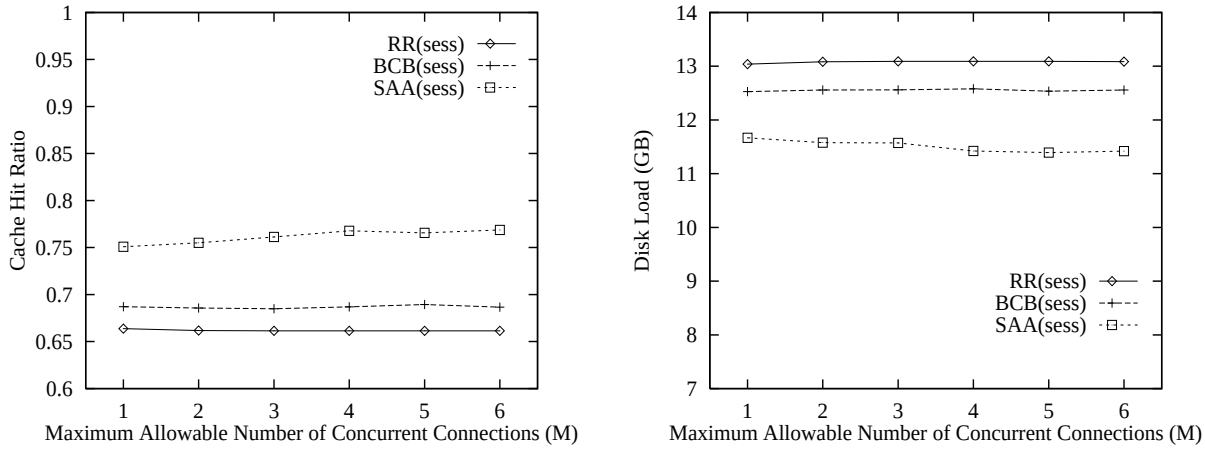
(a) BCB(sess) (b) SAA(sess)

Figure 18: Distribution of Request Concentration

5.5 Impact of Concurrent Connections

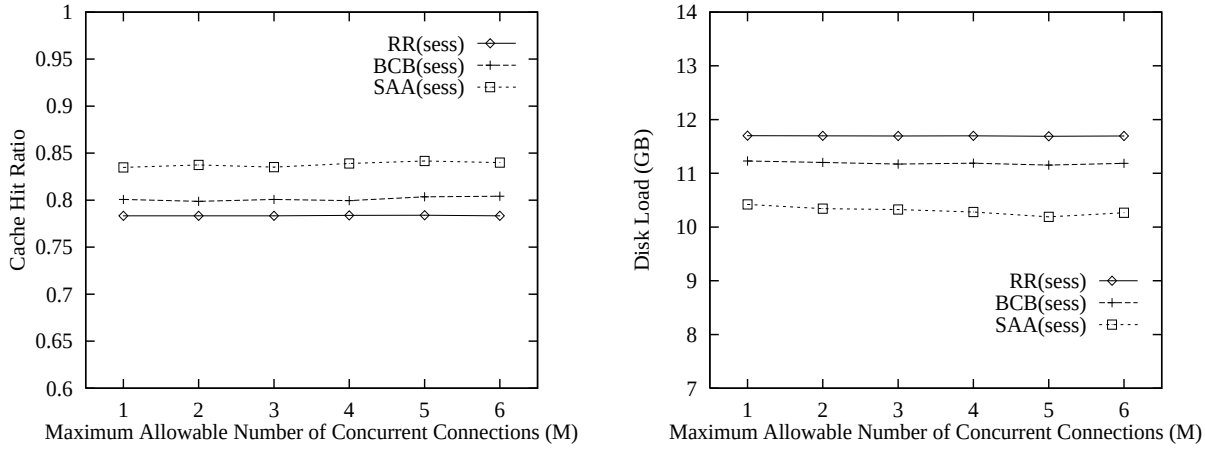
Some clients may set up a number of concurrent persistent connections to the web cluster to speed up retrieval [9]. Since the traces are dominated by HTTP/1.0 traffic, we used the following simple heuristic to decide when a new connection would be established for a request. When a request R arrives, a new connection is set up if and only if: (1) the number of active connections opened by the client is fewer

than the maximum allowable number M; and (2) the last request sent over each existing connection is within 1 second of R's arrival. Otherwise, the request is assigned to the existing connection whose last request is the least recent. With this heuristic, the number of concurrent connections set up by a client is largely related to its request rate — the higher the rate, the more the active connections. Figures 19 and 20 show the experimental results[5] for values of M between 1 and 6. In this set of experiments, the connection holding time was set at 15 seconds.



(a) Cache Hit Ratio (b) Disk Load

Figure 19: Impact of Concurrent Connections under LRU replacement



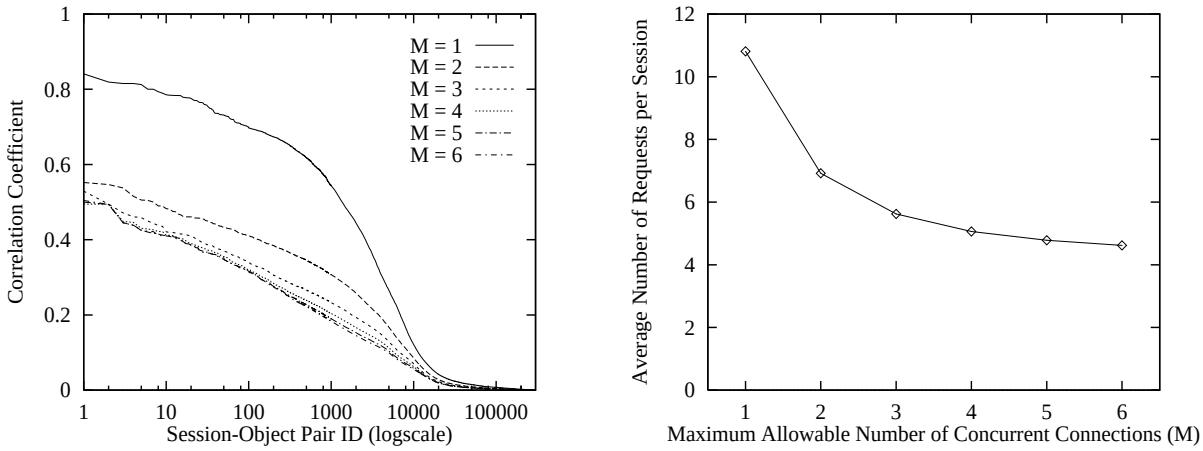
(a) Cache Hit Ratio (b) Disk Load

Figure 20: Impact of Concurrent Connections under GLFU replacement

As can be seen, the caching performance of the three algorithms is not significantly affected by the use of concurrent connections. The trend of RR(sess) is intuitive — due to the content-blind nature, the performance of RR(sess) is independent of connection management. Concurrent connections affect the performance of BCB(sess) and SAA(sess) in two aspects. First, setting up multiple

[5]We found that the value of M had little impact on the composition of sessions beyond 6.

connections by a single client reduces the predictability of requests on each connection. Figure 21(a) shows the correlation coefficients (see Section 3.3 for definition) of the WC/06/03 trace with different maximum numbers of concurrent connections. It is seen that the correlation coefficients decrease with increasing number of concurrent connections. This implies subsequent requests in the same session are less predictable given only the first request, thereby reducing caching effectiveness. On the other hand, a higher number of concurrent connections results in a larger number of sessions. As shown in Figure 21(b), this reduces the average number of requests in a session and leads to lower allocation dependency. Due to the combination of the two opposite effects described above, the performance of BCB(sess) and SAA(sess) does not vary much over the range of concurrent connections we examined (see Figures 19 and 20).



(a) Correlation Coefficients

(b) Avg. No. of Requests per Session

Figure 21: Two Different Impacts of Concurrent Connections

6 Conclusion

We have studied the caching performance of session-grained and request-grained allocations for web clusters under persistent connections. To investigate the best achievable caching performance under session-grained allocation, we have formulated the allocation problem as an optimization problem and presented a session-affinity aware algorithm based on a greedy heuristic. The new algorithm takes into consideration the probabilities of accessing different objects in the session. Through trace-driven simulation experiments with two different cache replacement algorithms, it is shown that: (1) at the request-grained level, content-based allocation exhibits much better caching performance than content-blind allocation; (2) at the session-grained level, however, allocation dependency offsets much of the performance benefits of content-based allocation, and the performance loss increases with cluster size and connection holding time; (3) the caching performance of session-grained allocation

can be improved by exploiting the correlation between the first and subsequent requests in a session; (4) optimizing session-grained allocation cannot fully compensate for the performance loss caused by allocation dependency.

References

[1] Thomas T. Kwan, Robert E. McGrath, and Daniel A. Reed. NCSA's world wide web server: Design and performance. *IEEE Computer*, 28(11):68–74, November 1995.

[2] Valeria Cardellini, Michele Colajanni, and Philip S. Yu. Dynamic load balancing on web-server systems. *IEEE Internet Computing*, 3(3):28–39, May/June 1999.

[3] Trevor Schroeder, Steve Goddard, and Byrav Ramamurthy. Scalable web server clustering technologies. *IEEE Network*, 14(3):38–45, May/June 2000.

[4] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich Nahum. Locality-aware request distribution in cluster-based network servers. In *Proceedings of the 8th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 205–216, October 1998.

[5] Richard B. Bunt, Derek L. Eager, Gregory M. Oster, and Carey L. Williamson. Achieving load balance and effective caching in clustered web servers. In *Proceedings of the 4th International Web Caching Workshop (WCW)*, April 1999.

[6] Ronald P. Doyle, Jeffrey S. Chase, Syam Gadde, and Amin M. Vahdat. The trickle-down effect: Web caching and server request distribution. In *Proceedings of the 6th International Web Caching and Content Delivery Workshop (WCW)*, June 2001.

[7] Valeria Cardellini, Emiliano Casalicchio, Michele Colajanni, and Philip S. Yu. The state of the art in locally distributed web-server systems. *ACM Computing Surveys*, 34(2):263–311, June 2002.

[8] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – HTTP/1.0. *RFC 1945*, May 1996.

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. *RFC 2068*, January 1997.

[10] Balachander Krishnamurthy and Jennifer Rexford. En passant: Predicting HTTP/1.1 traffic. In *Proceedings of IEEE GLOBECOM'99*, pages 1768–1773, December 1999.

[11] Balachander Krishnamurthy and Jennifer Rexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley, 2001.

[12] Jeffrey C. Mogul. The case for persistent-connection HTTP. In *Proceedings of ACM SIGCOMM'95*, pages 299–313, August 1995.

[13] Jeffrey S. Chase. Server switching: Yesterday and tomorrow. In *Proceedings of the 2001 IEEE Workshop on Internet Applications*, pages 114–123, July 2001.

[14] Michael Rabinovich and Oliver Spatscheck. *Web Caching and Replication*. Addison-Wesley, 2002.

[15] Ariel Cohen, Sampath Rangarajan, and Hamilton Slye. On the performance of TCP splicing for URL-aware redirection. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 117–125, October 1999.

[16] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Efficient support for P-HTTP in cluster-based web servers. In *Proceedings of the 1999 USENIX Annual Technical Conference*, pages 185–198, June 1999.

[17] Daniel M. Dias, William Kish, Rajat Mukherjee, and Renu Tewari. A scalable and highly available web server. In *Proceedings of the 41st IEEE Computer Society International Conference (COMPCON)*, pages 85–92, February 1996.

[18] Martin F. Arlitt and Carey L. Williamson. Internet web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–645, October 1997.

[19] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[20] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, March 1969.

[21] WorldCup98 web site access logs, 1998. http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[22] Martin Arlitt and Tai Jin. A workload characterization study of the 1998 world cup web site. *IEEE Network*, 14(3):30–37, May/June 2000.

[23] Apache HTTP server. http://www.apache.org.

[24] Charu Aggarwal, Joel L. Wolf, and Philip S. Yu. Caching on the world wide web. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):94–107, January/February 1999.

[25] Xueyan Tang and Samuel T. Chanson. Coordinated en-route web caching. *IEEE Transactions on Computers*, 51(6):595–607, June 2002.

[26] Junho Shim, Peter Scheuermann, and Radek Vingralek. Proxy cache algorithms: Design, implementation, and performance. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):549–562, July/August 1999.