Effectiveness of Load Balancing in a Distributed Web Caching System

Brandon Plumley, Richard Hurley

Department of Computing and Information Systems

Trent University

Peterborough, ON Canada

bplumley@trentu.ca, rhurley@trentu.ca

Abstract

In this paper, we investigate the effects of load balancing in a distributed Web caching system. Our investigation is focused specifically on adaptive load sharing: an approach that reacts to the current state of the system. Load balancing has been shown to improve system performance in other applications and in this paper, we investigate it in a distributed Web caching environment using both a unified and partitioned approach. The goal of this work is threefold: (1) to determine the conditions under which load balancing can be beneficial in a distributed Web caching system, (2) to compare load balancing in a unified and partitioned Web caching system, and (3) to determine how much state information is required to achieve any benefit. Discrete-event simulation is used as the tool to generate results for these different environments.

Keywords: Web Caching, Load balancing, Performance Evaluation, Simulation, Computer Modelling

1. Introduction

Web caching is a technique that is heavily utilized on the Internet and has been shown to be highly effective in improving network performance by reducing bandwidth and latency [1][2][3]. The premise of Web caching is to store frequently-accessed pages from an originating server *closer* to the clients to reduce bandwidth and workload on the originating server[4]. This can result in a reduction in the time to deliver a page from the server to the client [5].

One of the more common approaches is to implement multiple Web caches in a distributed system where additional Web caches are considered peers with each cache being contained within the same level (similar ”distance” from the client). This arrangement allows the peer caches to be relatively *close* to one another. Distributive Web caching allows for better load sharing when compared to other approaches [6].

Traditionally Web caches hold both large and small pages together, where one large page would replace multiple small pages or a single large page. This storage model is referred to as Unified caching. However, partitioned Web caching, where large and small pages are stored in separate

areas, has been shown in previous work to result in increased performance [7]. This approach ensures that large pages will not replace many small pages in the cache.

Since there are multiple caches working together in a distributed environment, a key mechanism to fully harness the potential of the system is load balancing. There have been many load balancing algorithms proposed in the past that have been applied to diverse applications such as telecommunications, processing process on a computer and network traffic [8][9]. In the last decade there has been an increase into research for applying load balancing to a distributed Web caching system [10][11]. In a system without load balancing, requests are typically assigned randomly to the distributed Web caches. With no direction as to which assignment of requests, the issue that arises is that one cache could be congested while other caches are underutilized; this uneven utilization can degrade performance [12].

Typically, there are two common transfer policies used in adaptive load sharing: sender-initiated and receiver-initiated [13]. A sender-initiated policy attempts to balance the workload in the system at the point in time when a Web cache receives an incoming request. A threshold value, which is based on the number of requests in the local queue, is used to determine whether the system needs to transfer the incoming request to another peer cache. This approach will only transfer newly arriving requests with them being placed at the end of the selected Web cache queue. A receiver-initiated policy, on the other hand, attempts to load balance as requests are serviced (not when they arrive). If the Web cache queue falls below a given threshold, the system attempts to find additional work from a peer cache with queue length above the given threshold. If such a cache can be found, a request from its tail will be transferred to the tail of the Web cache that initiated the transfer. In this paper, we focus on sender-initiated policies but more information on the performance of receiver-initiated policies can be found in [14].

For this paper, we used a discrete-event simulation model to investigate the performance of load balancing in a distributed unified and partitioned Web caching system. We present the models and assumptions for our distributed Web caching systems for both unified and partitioned storage in Section 2, while in Section 3 the input parameters are discussed. Section 4 presents the simulation results derived from the models and finally, Section 5 summarizes our findings.

2. Performance Models

Our system model is divided into two parts: a Web reference model and a Web cache model. By varying the architecture in the Web cache model, we produce two distinct systems: unified and partitioned. We can simplify our system models since we are concerned with the relative performance achieved by each load balancing algorithm relative to the distributed Web caching system without load balancing (i.e. we are not concerned with the absolute performance of the system).

2.1. Web Reference Model

The pages stored in a Web cache and their request probabilities vary over time. Pages such as news articles, viral videos, course assignments and memes become popular for periods of time and then eventually the frequency of access decreases. To represent this behavior, we use a dynamic page reference model (shown in Figure 1) as described in [15].

2.1.1. Page Popularity

The request probabilities are shown in Equation (1) and defined by: $p_i(t)$, the probability of requesting page i at time t is $i = 1, 2, ..., M$, where M is the number of Web pages. From Figure 1, we can see that there are two states for the probability of requesting a page: normal and popular. Pages in the popular state have a higher request probability than that of the pages in the normal state, where v represents the ratio of the rate of requests in the popular to normal state.

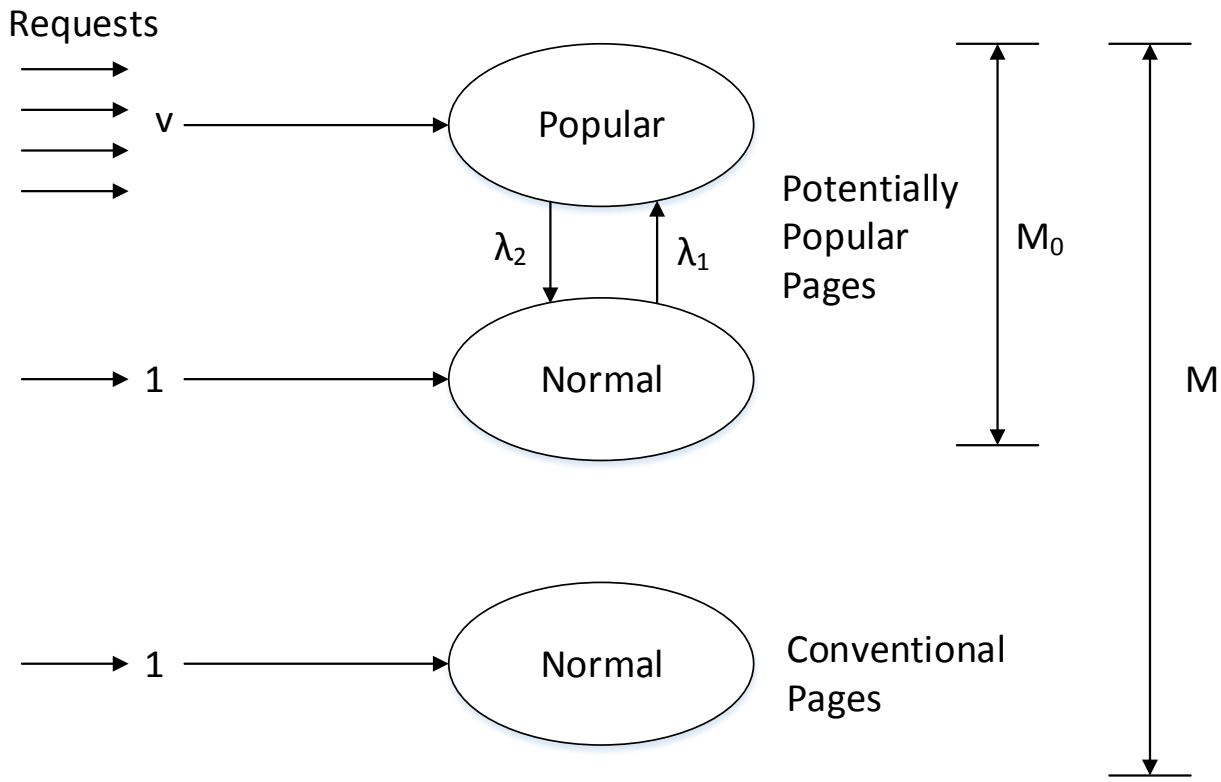

Figure 1: Dynamic Page Reference Model

The model also assumes that there are two types of pages: conventional (M) and potentially popular (M_0). Conventional pages remain in the normal state while potentially popular pages shift between the normal and popular sate based on a continuous-time Markov chain. The rate at which a page transitions from a normal to popular state is λ_1 and from popular to normal is λ_2 (the time spent in either state is assumed to be exponentially distributed). We let $M_0 < M$ denote the number of potentially popular pages and thus $M_p(t) < M_0$ represents the total number of pages in the popular state at time t. The time-dependent request probability for page i is defined as:

$$p_i(t) = \begin{cases} \frac{v}{vM_p(t)+(M-M_p(t))} & \text{popular state} \\ \frac{1}{vM_p(t)+(M-M_p(t))} & \text{normal state} \end{cases} \quad (1)$$

2.1.2. Page Size

To simplify our model we assume that a page is either large or small, we assume that a large page is k times larger than a small page. Small pages have a service time that is assumed to be exponentially distributed with a mean rate of μ^{-1}, while large pages have a exponentially distributed service time of $k\mu^{-1}$.

It has been shown that the majority (ninety percent) of Web pages are in the range of 100 bytes to 100 KB, with less than ten percent being greater than 100 KB [16]. As a result, we assume that the probability of requesting a large page would be $1 - s$, where s is the probability of requesting a small page. Since 90% of pages requested are small, we set s to 0.9, which based on previous observations is reasonable.

2.2. Web Cache Model

Our Web cache model is comprised of a page replacement model, an architectural model, and a storage mode.

2.2.1. Page Replacement Model

One of the most critical components of a good Web caching system is the page replacement algorithm. The page replacement algorithm is responsible for storing or discarding pages in the Web cache once it becomes full. Without this component, once the cache is full, no new pages would be stored and the cache would become stale. Although there are many different page replacement algorithms, our model uses The Least Recently Used (LRU) [17].

As the name implies, the LRU algorithm selects the least recently used page (determined from the last accessed timestamp) to be removed from the cache. We have chosen to implement the LRU since it is one of the most widely-used cache replacement algorithms for Web pages [18]. One of the main advantages of the LRU is that it is straightforward to incorporate in the system model, while being highly efficient. Some of the determents to the algorithm are that it excludes certain state information such as the and latency of a page. However, since we are considering only relative performance, these effects will be negligible.

2.2.2. Architectural Model

Our work expands on a Web cache model that was first introduced by [19], and is shown in Figure 2. The distributed system contains D peer (or co-operative) caches which are assumed to exist as the same level. When a Web cache receives a page request, the cache fist checks if there is a copy currently stored in its own cache and if the page is found, it is returned to the client. However, if no copy of the requested page can be found at the current Web cache, the request is forwarded randomly to one of the peer caches. If the page cannot be found at the new Web cache, the request is again transfered to another peer Web cache, until the page is found. If all D peer caches are exhausted the request will be forwarded to the originating server, a copy is made at the original cache and the page is returned to the client.

It is assumed that if the request is satisfied by the first cache in D peer cache, than the processing time is considered to be T_0 (this includes the service time and propagation delay). If the first cache

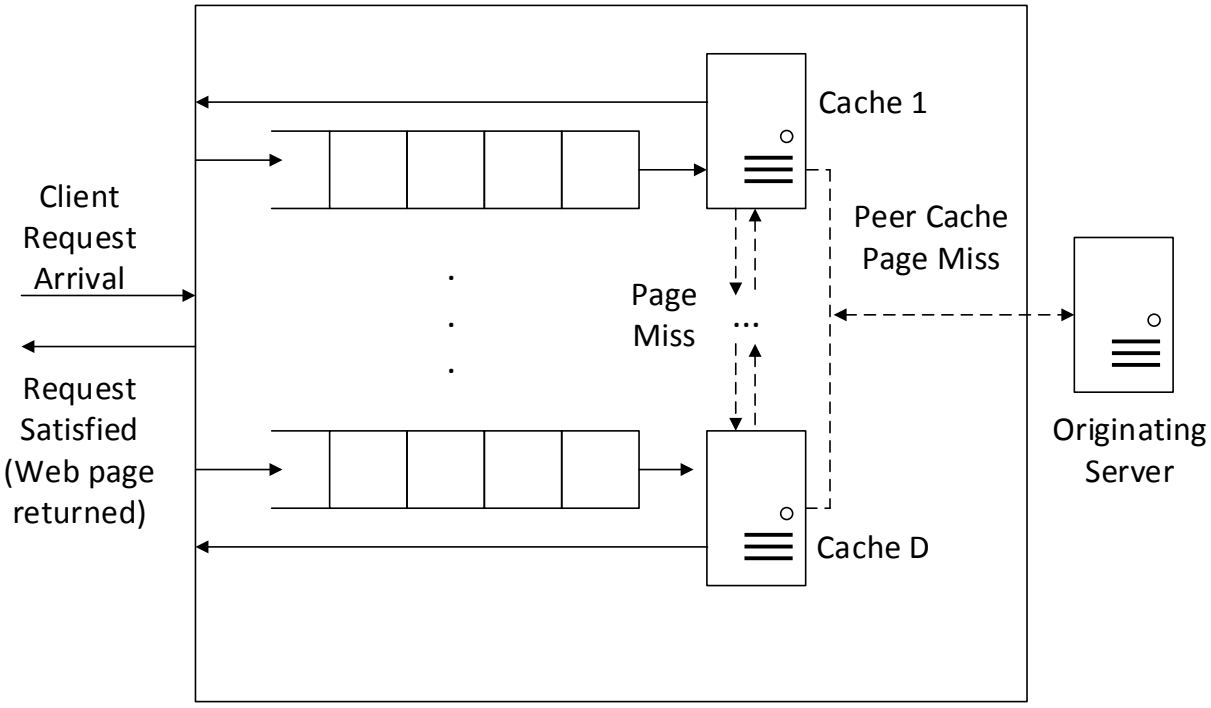

Figure 2: Web Caching Architecture Model

can not satisfy the request (a {textitmiss) but can still be satisfied within the D peer cache, then the service time is assumed to be $T_1 + T_0$. While exact for $D = 2$ caches, this value is an approximation for larger values of D as it would be a factor of the number of cache misses. If the request can not be satisfied within our distributed Web caching system and therefore must be completed by the originating server, then the processing time is considered to be T_2.

2.2.3. Storage Model

We also investigate two variations of the cache storage model: a unified cache and a partitioned cache. A unified cache is simply a single cache that treats both large and small pages the same (they are stored together). If the cache was full and needed to make room for a incoming large page, the cache would have to discard one large page or k small pages. A partitioned cache on the other hand treats large and small pages differently. The cache is split into two separate ares: one for large pages and one for small pages. This approach ensures that large pages will not replace k small pages and that k small pages will not replace a single large page. It is assumed that the ratio of space reserved for large pages is (P_L).

2.3. Load Balancing Algorithms

Our investigation considers two variants of a sender-initiated load balancing algorithm:

- *Short-Sender (SS).* Once a threshold value (Θ) is reached, the algorithm looks for the Web cache with the shortest queue (including itself).

- *Random-Sender (RS).* Once a threshold value (Θ) is reached, the algorithm randomly selects a Web cache (excluding itself).

3. Input Parameters

In order to simplify our investigation, the following model parameters are fixed for all simulations: $M = 1000$, $M_0 = 100$, $\mu = 1$, $P_L = 0.4$, $T_0 = 0.5$, $T_1 = 0.1$, $T_2 = 1$, $s = 0.9$, $z = 100$ and $k = 10$. It is assumed that our system has a finite population of N client workstations, with D peer caches. Each Web cache is assumed to have a size of C bytes, which is defined to be the percentage of total bytes available for storage within the entire system, initially we set C to 0.05 [19].

4. Performance Results

The main objective of this investigation is to evaluate the relative performance of our load balancing algorithms in a distributed Web caching system using both unified and partitioned storage against the same system without load balancing. That is, the chief concern is whether load balancing will be effective in a distributed Web caching system. We are not concerned with the absolute performance of our system but that said, it would be beneficial to also be able to compare the results from the simluation models with experimental data from an implemented system but at this point in time, none was available. This is an area underwhich current work is being applied. Our performance measure of interest in our simulation models is mean response time (the time from when a request is generated until the web page has been returned to the client). The complexity of the system and the number of possible parameters is such that an analytic solution is not tractable thus results are gathered using a discrete event simulation written in C++. For more information on the acutal simluation program, please see [14].

4.1. Threshold Limit

We begin by examining threshold limit (Θ) for a sender-initiated approach in a unified and partitioned storage environment. We simulate the system under a high system load ($\rho_{N_U} = 0.85$) for $D = 2$ and 10 peer Web caches (Figures 3 and 4). The results indicate that all four systems (RS_U - Random-Sender-Unified, RS_P - Random-Sender-Partitioned, SS_U - Short-Sender-Unified, SS_P - Short-Sender-Partitioned) preform at least as well as to that of the distributed Web caching system without load balancing (N_U - No LB-Unified, $N_P - NoLB - Partitioned$). In some cases, response time is decreased by as much as 60.0%.

One of the more prominent trends is that as we increase Θ, the mean response time also increases: this is as a result of the fact that less load balancing is occurring up until the point where no pages are being transfered. However, with the RS_U algorithm we observe a small dip: the valley of the dip tends to be achieved with a threshold value (Θ) just greater than 0 (1 or 2). This can be explained by the fact that when the threshold is set to 0, the system will transfer the work randomly even if the local Web cache queue is the shortest. As the threshold is increased, the probability that the arriving Web cache is the shortest in the system decreases. As the threshold is increased to the point where the algorithm stops initiating transfers, there appears to be an *optimal* value that would be dependent on factors such as system load and number of caches. Going forward, we will be using a threshold value (Θ) of 3, as this is a reasonable choice given that the optimal value can not be directly determined.

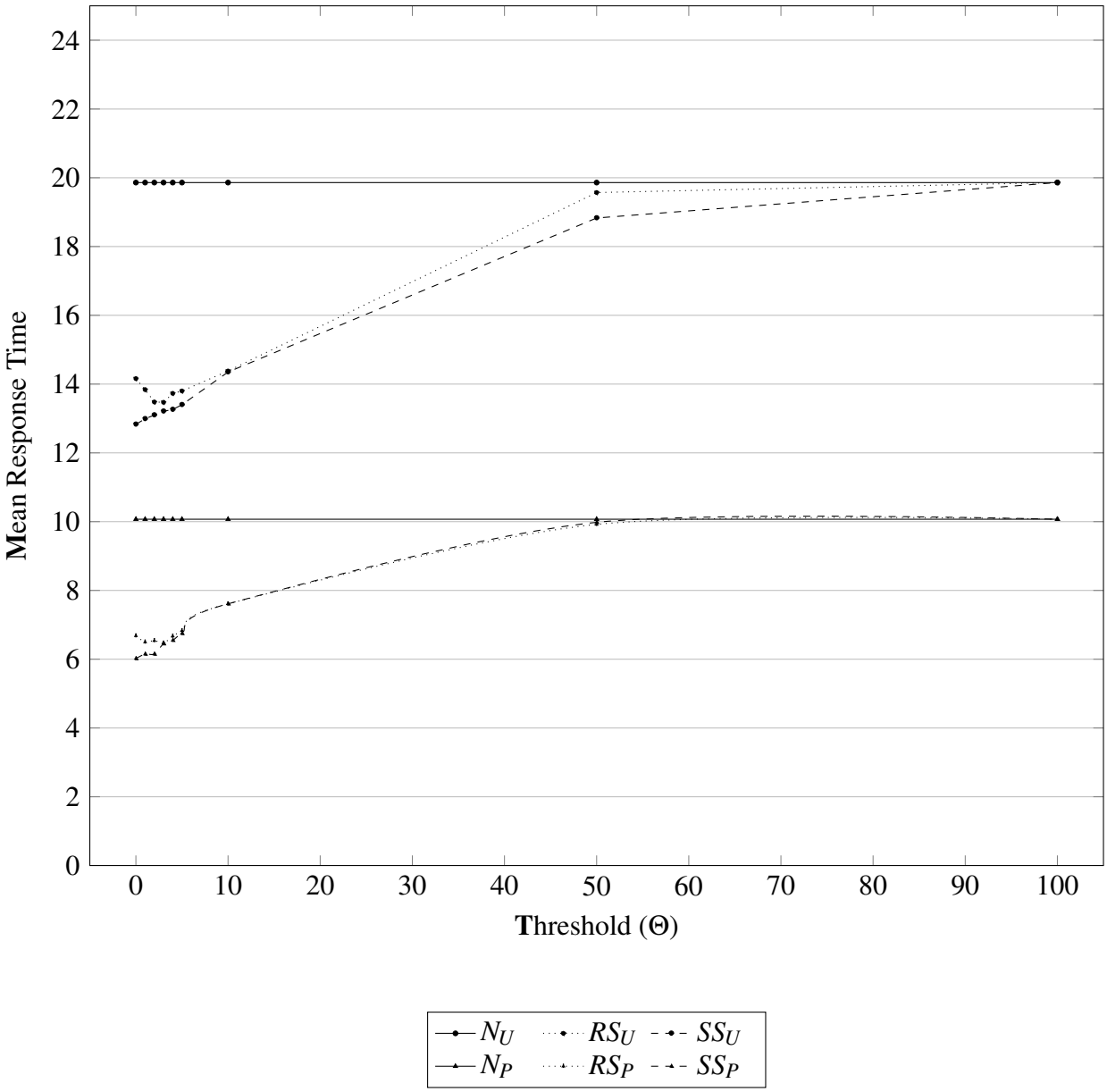

Figure 3: The Effect of Threshold (Θ) on the Mean Response Time for Unified and Partitioned Web Caching: $D = 2$, $\rho_{N_U} \approx 0.85$

4.2. System Workload

We next examine the effects of system workload for $D = 2$ and 10 peer Web caches (Figures 5 and 6). We observe that the system is relatively underutilized (workloads less than 20%), there is little difference between the load balancing algorithms and the respective systems without load balancing. As the utilization increases, we start to see a dramatic improvement (with respect to response time) with our load balancing algorithms relative to the systems without load balancing: this trend becomes more noticeable as the number of peer caches increase. Specifically from Figure 6, the Short (SS_U) algorithm has a decrease in response time of 37.7% over N_U ($\rho_{N_U} \approx 90\%$), while RS_U has a decrease in mean response time of 30.6% over N_U ($\rho_{N_U} \approx 90\%$). The Short

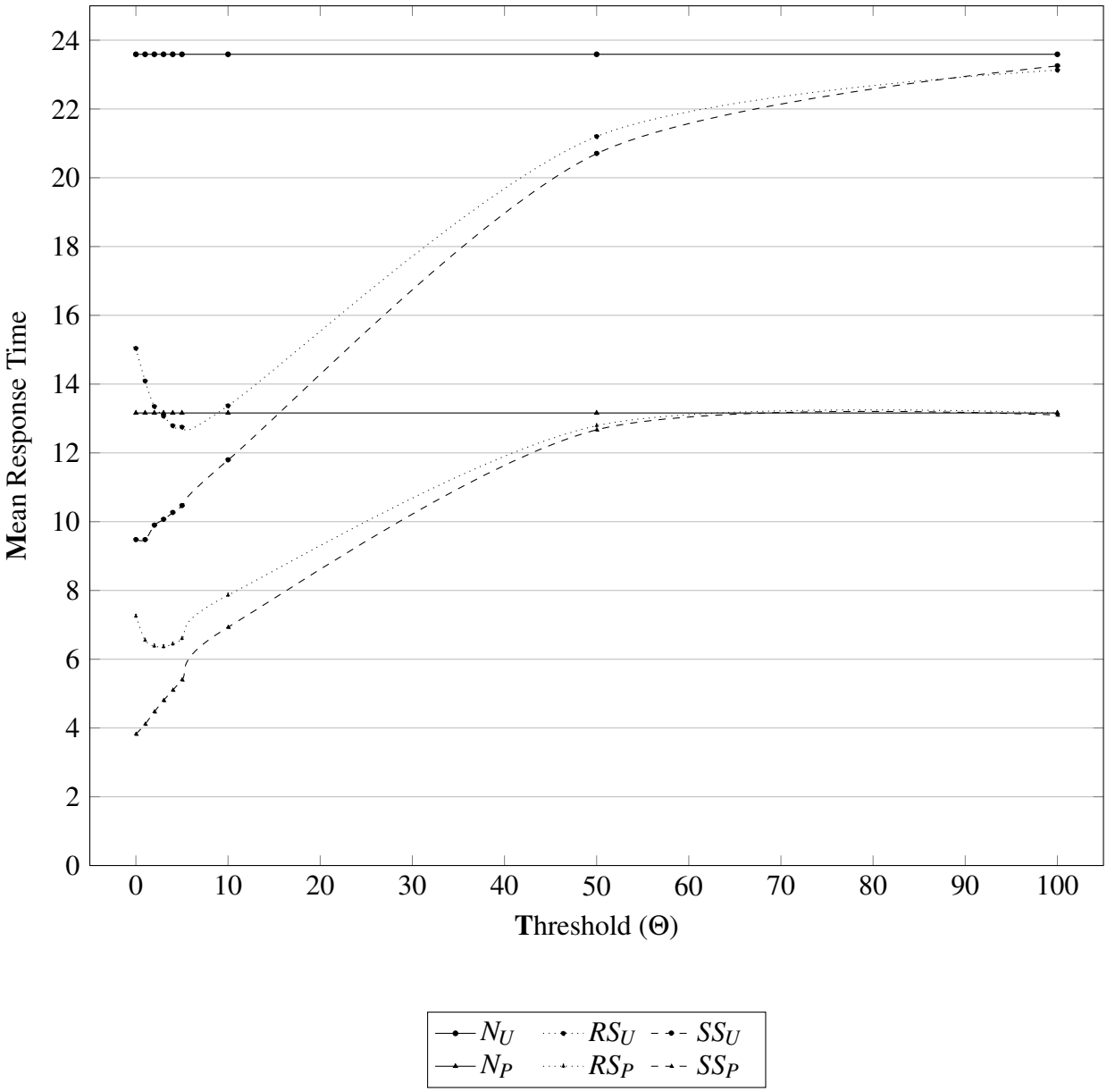

Figure 4: The Effect of Threshold (Θ) on the Mean Response Time for Unified and Partitioned Web Caching: $D = 10$, $\rho_{N_U} \approx 0.85$

(SS_U) algorithm seems to outperform the Random (RS_U) algorithm by 7.1% ($\rho_{N_U} \approx 90\%$). The results also indicate that partitioned load balancing systems follow the same trends as their unified counterparts with partitioning tending to perform better overall.

Additional workload seems to provide more opportunity for the load balancing algorithms to reduce the mean response time and so we can conclude that the higher the system load, the more potential the load balancing algorithms have to make a positive impact on the performance of the distributed Web caching system in both a unified and partitioned storage model.

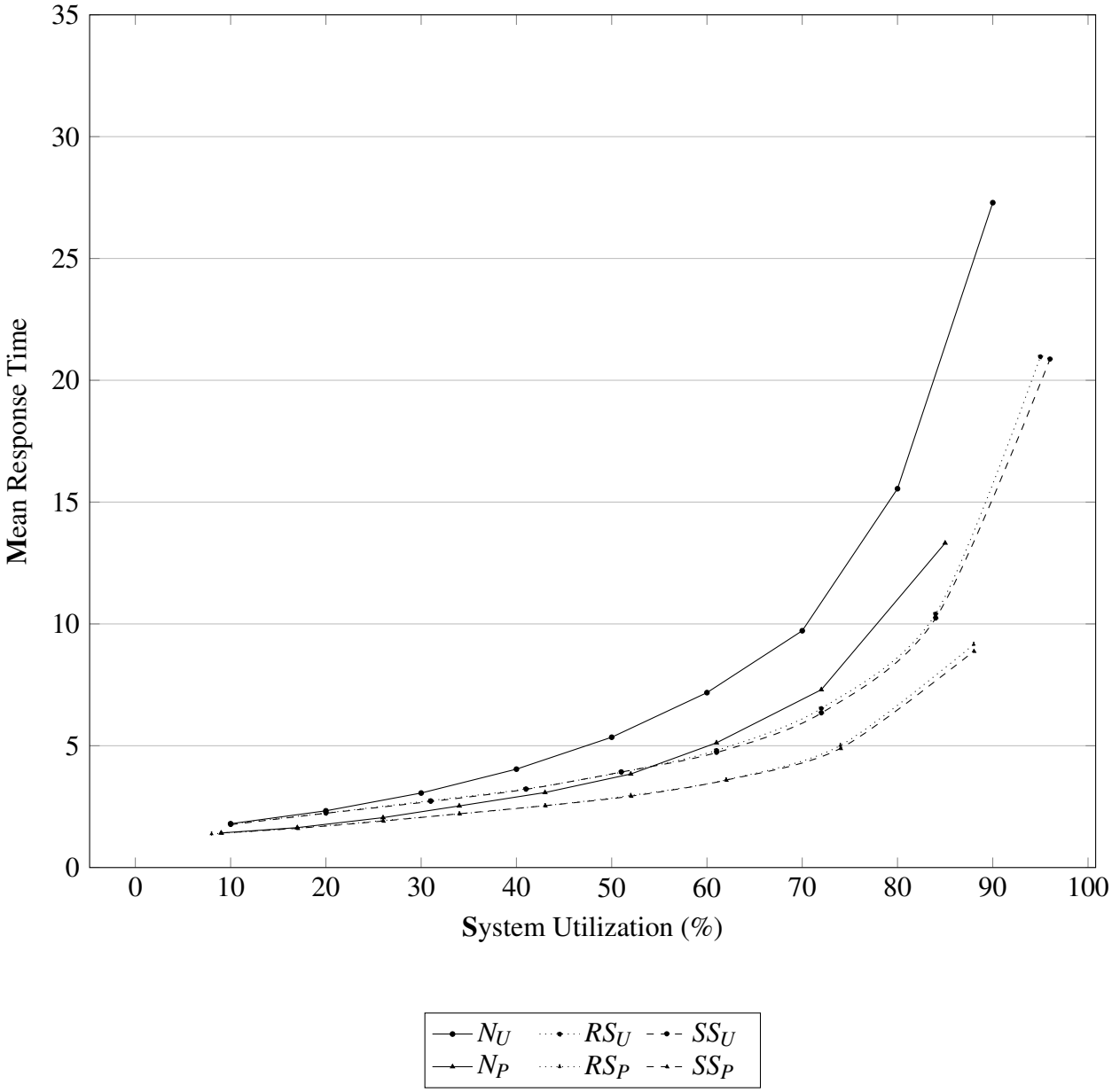

Figure 5: The Effect of System Workload on the Mean Response Time for Unified and Partitioned Web Caching
$\Theta = 3$, $D = 2$

4.3. Number of Peer Caches

We examine both Web caching systems under a medium (Figure 7) and a high system load (Figure 8). As additional peer Web caches are added, the systems without load balancing (N_U, N_P) have mean response times which tend to increase marginally. From Figure 8, the increase in mean response time from 2 to 10 peer Web caches for the systems without load balancing (N_U and N_P) N_U is 12.0% and N_P respectively. Each additional Web cache added to the distributed Web caching system tend to increase the probability that one of the Web caches will become overloaded, leading

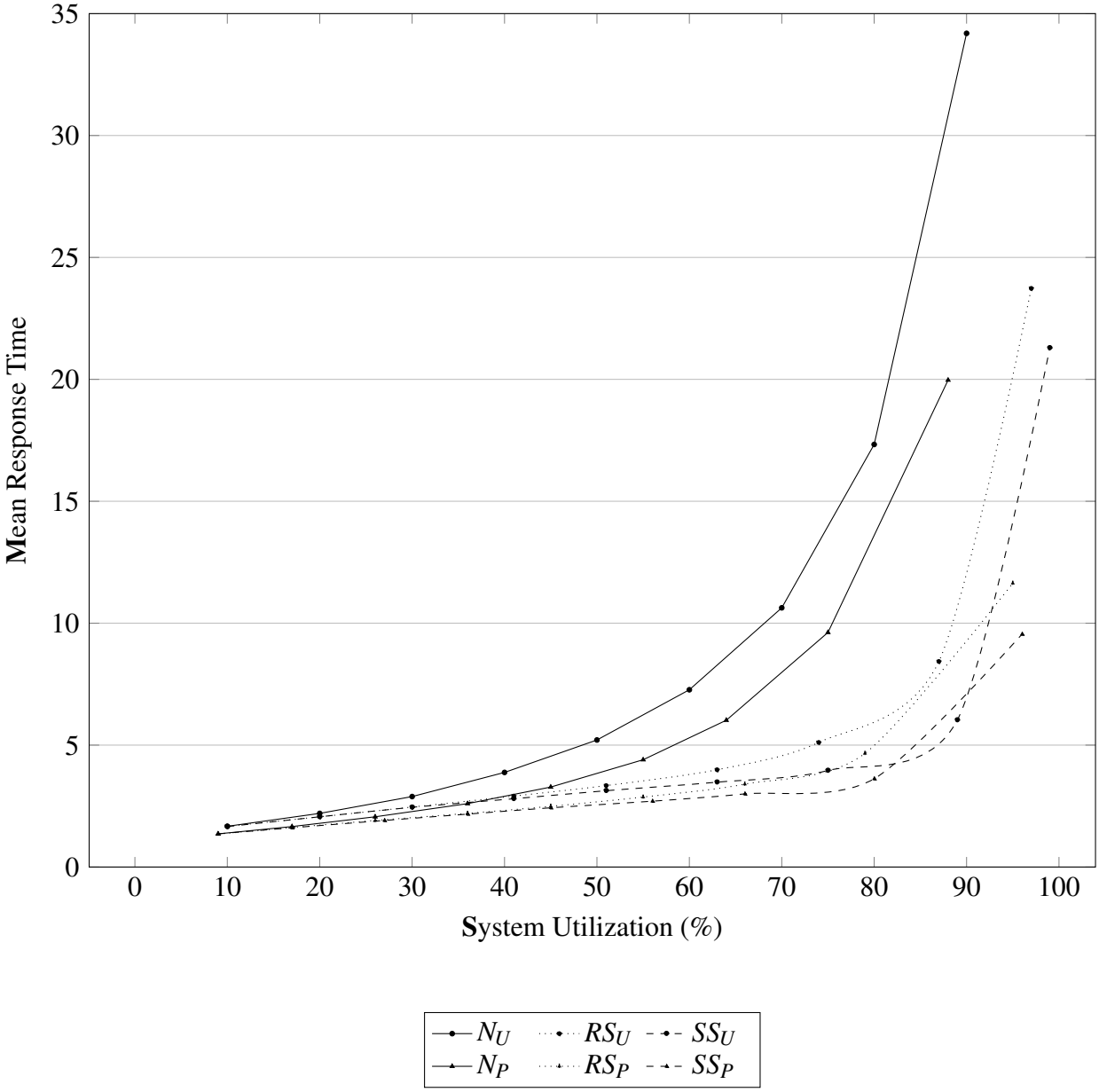

Figure 6: The Effect of System Workload on the Mean Response Time for Unified and Partitioned Web Caching
$\Theta = 3$, $D = 10$

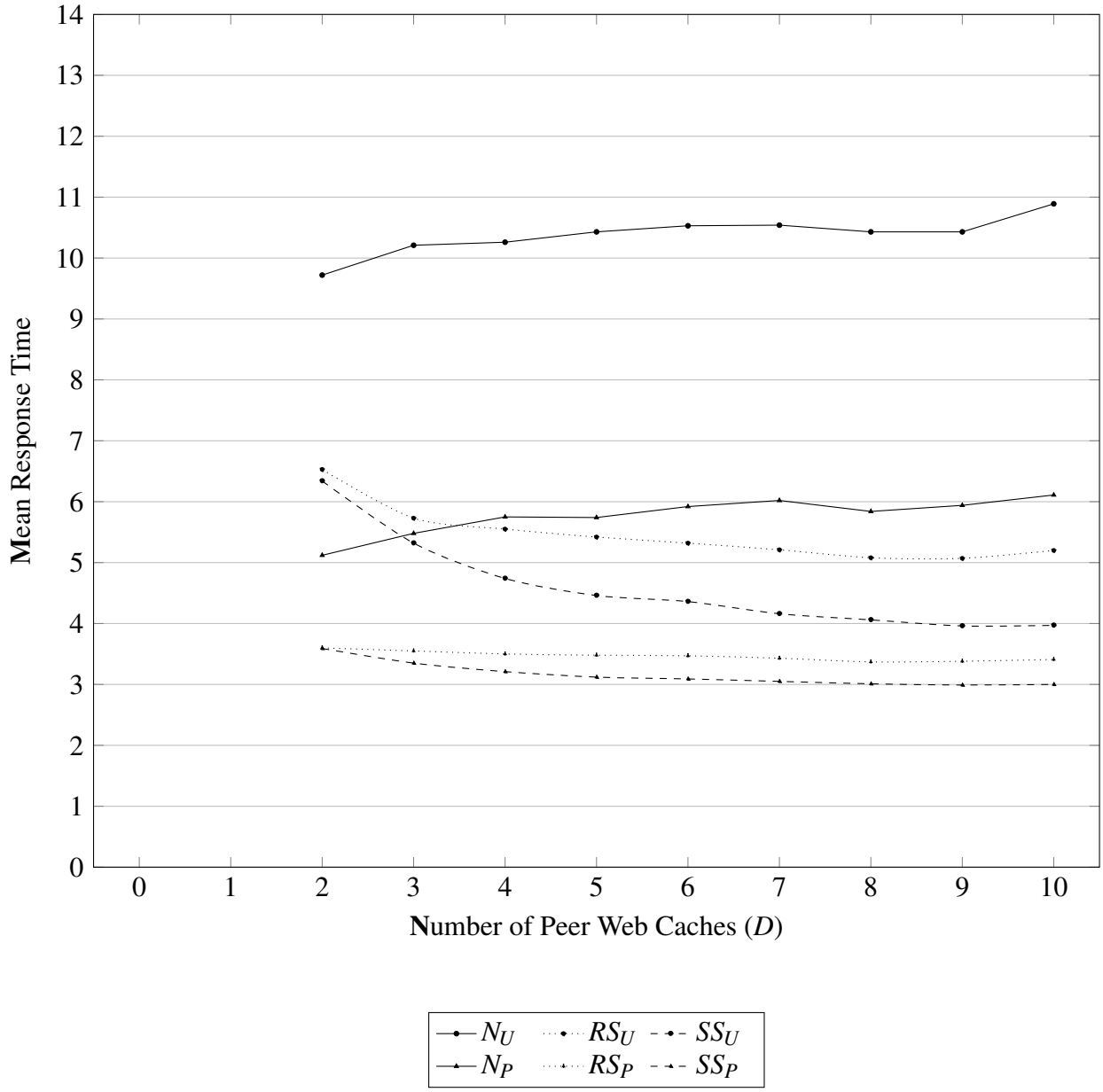

Figure 7: The Effect of the Number of Peer Caches (D) on the Mean Response Time for Unified and Partitioned Web Caching: $\Theta = 3$, $\rho_{N_U} \approx 0.70$

to higher response times.

As additional Web caches are introduced in our load balancing environments, the Short (SS_U, SS_P) and Random (RS_U, RS_P) algorithms tend to lead to a decrease in mean response time. For each additional cache added to the system, the system is provided with more opportunities to attempt to balance the workload in the system, leading to a decrease in mean response time. Again from Figure 8, when D = 10, SS_U and RS_U have a decrease in mean response time of 57.1% and 44.3% respectively with regards to the system without load balancing (N_U). The algorithms seem to follow the same pattern with the Short algorithm outperforming the Random algorithm by 30.0%

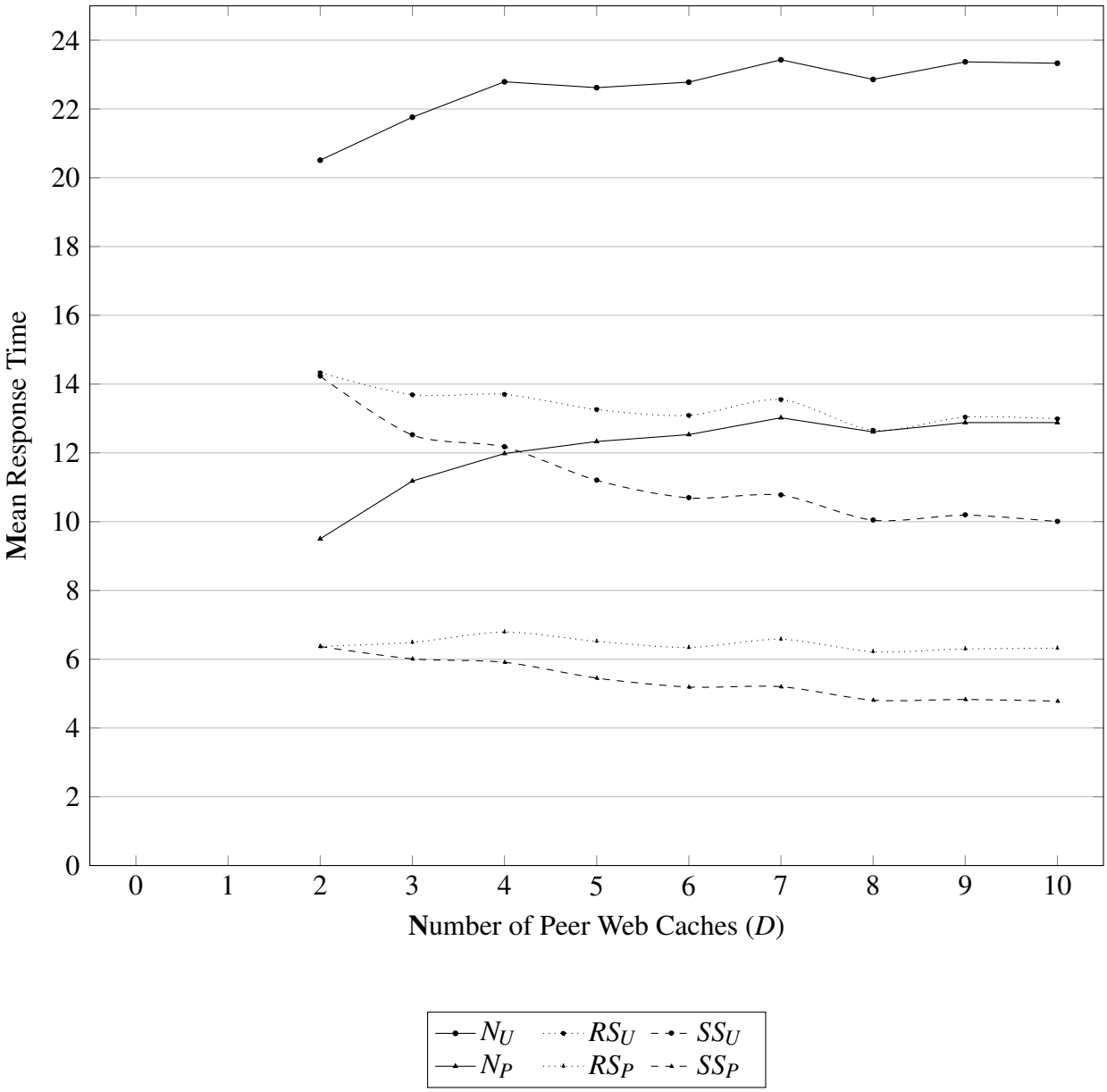

Figure 8: The Effect of the Number of Peer Caches (D) on the Mean Response Time for Unified and Partitioned Web Caching: $\Theta = 3$, $\rho_{N_U} \approx 0.85$

with respect to response time. However, it is important to note that Short algorithm would incur more overhead than Random algorithm due to the need to collect queue lengths from peer caches.

Partitioned Web caching system tends to again outperform a unified Web caching system. As we observe from Figure 8, there is a 52.2% performance difference between SS_U and SS_P and a 51.3% performance difference between RS_U and RS_P when $D = 10$. From these results, we conclude that a Web caching system with load balancing tends to scale gracefully relative to a Web caching system without load balancing as the number of peer Web caches (D) increases.

4.4. Cache Size

In Figure 9, we investigate the effects of the cache size (C) on the mean response time for our unified and partitioned Web caching systems. As expected, we observe that there is a dramatic decrease in the mean response time when the Web cache size is greater than 0. As soon as Web caching is introduced, there is an immediate performance benefit that can be observed, N_U has a performance increase of 62.9% when $C = 5\%$ when compared to when $C = 0\%$. As the Web cache size increases (C), the mean response time tends to decrease. However, we observe that after the initial dramatic decrease in mean response time, the system does not see the same large performance benefit as the cache size continues to increase. The performance improvements over time tends to decrease until mean response time plateaus. This occurs when the cache size is large enough to store most of the pages from the originating servers (an unlikely event but does provide a lower bound for the mean response time). Both algorithms follow the same trend and tend to outperform the systems without load balancing. For example SS_U and SS_P have a performance increase of 35.8% over N_U and N_P, while RS_U and RS_P have an increase of 33.7% over N_U and N_P when $C = 100$.

We also observe that as we increase the cache size, the partitioned system collapses into a unified system. With ample cache space, both storage models achieve the same level of performance. We find that smaller values of cache size (as long as it is greater than 0), tends to benefit partitioned storage over unified storage(i.e. when $C = 5\%$, the mean response time for N_P is 53.7% lower than that of N_U). It is again important to observe that irrespective of the value of the cache size, load balancing tends to improve the performance of the system with respect to the mean response time.

5. Conclusion

The results from this study have shown that the load balancing algorithms in a distributed Web caching system can be effective from a performance standpoint. In fact, any of the algorithms we examined achieved a level of performance equal to or better than a system without load balancing. We also determined that both unified and partitioned systems scale well with respect to additional peer Web caches, with the performance gains actually increasing as additional caches are introduced (unlike the system without load balancing (N_U) which degrades with additional Web caches). The use of the partitioned storage system has also been shown to increase the performance benefits of the load balancing algorithms in the Web caching environment. Performance benefits are seen even if a simple algorithm such as Random is incorporated. The benefits tend to increase with the use of state information (such as that seen with Short versus Random algorithms). In all of our cases, the use of load balancing in a distributed Web caching system tends to be much more desirable relative to a Web caching system without load balancing.

The research from this investigation has opened the door to a variety of potential extensions. A natural extension would be to utilize more state information from the requests; specifically, what page is being requested. For example, it may be beneficial to transfer a request (or multiple requests) to a Web cache that contains the requested page so as not to have to retrieve the page from the originating server. This will result in a cache hit for the local Web cache, thus increasing the hit rate of the cache at the same time as reducing the mean response time. As well, our system model was based on a distributed Web caching system, it may be possible to adapt our sender-initiated load balancing algorithms to a hierarchal Web caching system where caches are assumed to reside at various levels (i.e."distances") from the client (a receiver-initiated would not be appropriate for this

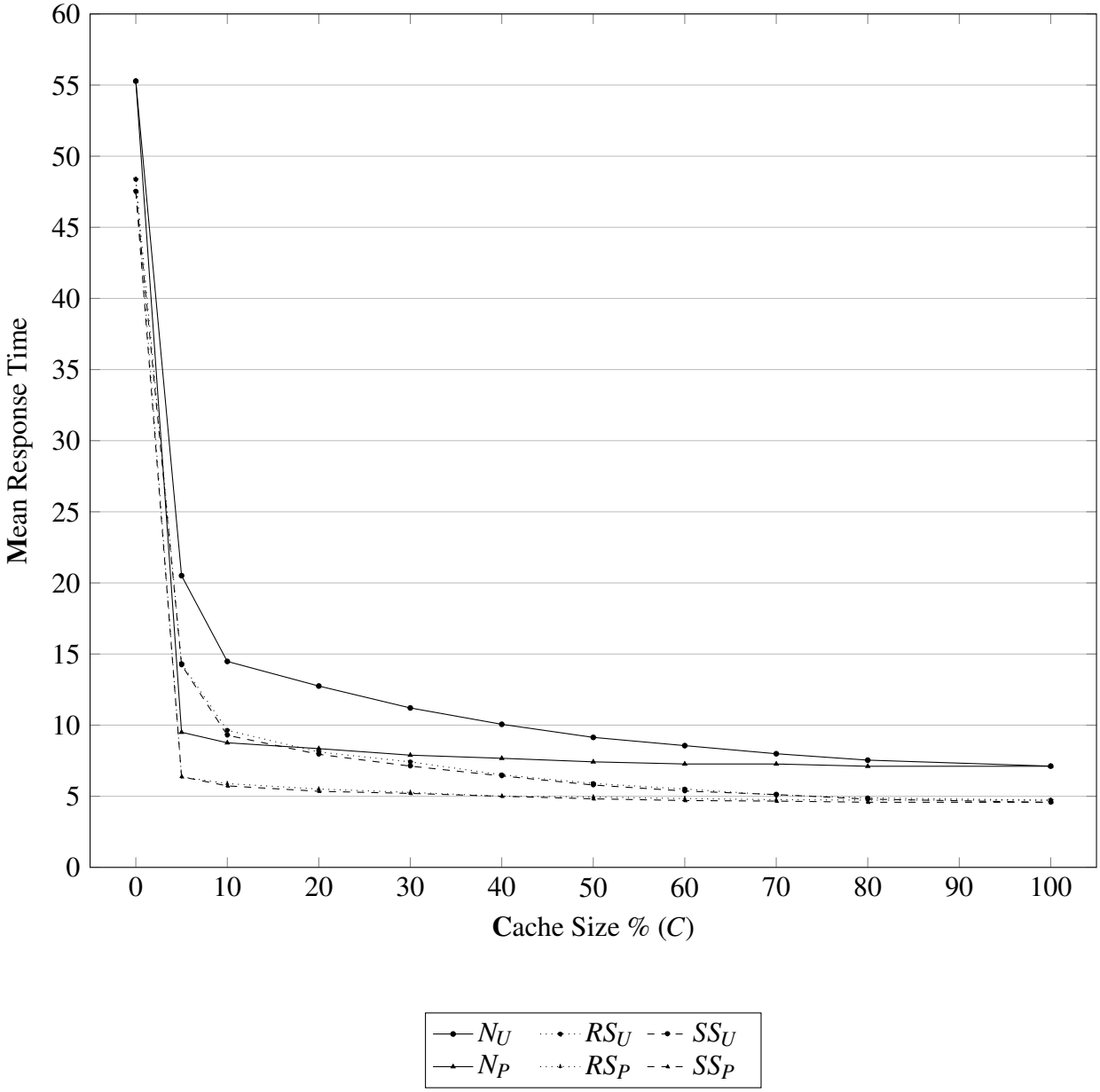

Figure 9: The Effect of Threshold Cache Size (C) on the Mean Response Time for Unified and Partitioned Web Caching: $\Theta = 3$, $D = 2$, $\rho_{N_U} \approx 0.85$

environment). Finally, our system model did not directly model the effects of overhead, such as the cost of transferring a request or the cost of collecting state information. It would be interesting to examine the effects of these overhead costs as they would likely impact some of the load balancing algorithms differently.

References

[1] W. Ali, S. M. Shamsuddin, and A. S. Ismail. A survey of web caching and prefetching. *Int. J. Advance. Soft Comput. Appl*, 3(1):18–44, 2011.

[2] Jay Chen and Lakshmi Subramanian. Interactive web caching for slow or intermittent networks. In *ACM DEV-4 '13: Proceedings of the 4th Annual Symposium on Computing for Development*, pages 1–10. ACM, 2013.

[3] Ali Raza amd Yasir Zaki, Thomas Pötsch, Jay Chen, and Lakshmi Subramanian. Extreme web caching for faster web browsing. In *SIGCOMM '15: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 111–112. ACM, 2015.

[4] Sunghwan Ihm and Vivek S. Pai. Towards understanding modern web traffic. In *ACM SIGMETRICS Performance Evaluation Review*, volume 39, pages 335–336. ACM, 2011.

[5] W. Feng, S. Man, and G. Hu. Markov tree prediction on web cache prefetching. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pages 105–120. Springer, 2009.

[6] R. T. Hurley, W. Feng, and B. Y. Li. Partitioning in distributed and hierarchical web-caching architectures: A performance comparison. *Proc. of the16th International Conference on Computer Applications in Industry and Engineering, Las Vegas, Nevada, USA*, pages 11–13, Nov 2003.

[7] W. Feng, R. T. Hurley, and Z. Tan. Increasing web cache hit rate by dynamic load partitioning. *Proceedings of the 7th Joint Conference on Information Sciences, Cary, North Carolina, USA*, pages 405–409, September 2003.

[8] D. Ferrari and S. Zhou. An empirical investigation of load indices for load balancing applications. Technical report, DTIC Document, 1987.

[9] S. Hofmeyr, C. Iancu, and F. Blagojević. Load balancing on speed. In *ACM Sigplan Notices*, volume 45, pages 147–158. ACM, 2010.

[10] G. Barish and K. Obraczke. World wide web caching: Trends and techniques. *IEEE Communications magazine*, 38(5):178–184, 2000.

[11] Rohan Gandhi, Hongqiang Harry Liu, Y. Charlie Hu, Guohan Lu, Jitendra Padhye, Lihua Yuan, and Ming Zhang. Duet: cloud scale load balancing with hardware and software. In *SIGCOMM '14: Proceedings of the 2014 ACM conference on SIGCOMM*, pages 27–38. ACM, 2014.

[12] M. E. Soklic. Simulation of load balancing algorithms. *ACM - SIGCSE Bulletin*, December 2002.

[13] D. L. Eager, E. D. Lazowsk, and J. Zahorjan. A comparison of receiver-initiated and sender-initiated adaptive load sharing. *ACM 0-89791-169-5/85/007/0001*, 1985.

[14] B. Plumley. An investigation of load balancing in a distributed web caching systems. Master's thesis, Trent University, 2015.

[15] R. T. Hurley and B. F. Hircock. Benefits of vertical file migration in a horizontal file migration system. *IASTED International Conference on Parallel and Distributed Computing Systems, MIT, Boston, MA, USA*, pages 365–370, Nov. 3-6 1999.

[16] M. F. Arlitt and C. L. Williamson. Web server workload characterization: The search for invariants. In *ACM SIGMETRICS Performance Evaluation Review*, volume 24, pages 126–137. ACM, 1996.

[17] H. Bahn, H. Lee, and S. H. Noh. Replica-aware caching for web proxies. *Computer Communications Journal, Elsevier Science*, 2001.

[18] R. T. Hurley and B. Y. Li. Effects of dynamic content on web caching. *ISCA PDCCS*, 2008.

[19] B. Y. Li. An investigation of partitioned caching in the world wide web. Master's thesis, Trent University, 2002.