An Adaptive Hierarchy Management System for Web Caches

Pranav A. Desai
Stratacache Networking
1031 E Third St.
Dayton, OH - 45429
pdesai@stratacache.com

Jaspal Subhlok
Department of Computer Science
University of Houston
Houston, TX - 77204
jaspal@cs.uh.edu

Abstract

A group of web caches can be organized into a cooperative hierarchy where a search for a requested object is performed among the cooperating peer caches before the object request is sent to the origin web server. Such cooperation improves the overall hit ratio but introduces the overhead of an additional step in request processing as well as additional workload on the caches and more traffic in the network. The main goal of cooperation is to improve the response time observed by clients, but the result can be the opposite, if, e.g., the network between the caches is very busy. In this paper, we present an Adaptive Hierarchy Management System to dynamically configure a hierarchy of caches based on existing conditions, without human intervention. The system consists of a centralized controller and distributed agents installed on the caches. The agents collect and report information on cache usage. The controller monitors the underlying network and determines the best hierarchy based on cache and network information. The new hierarchy is then broadcast to all agents. The agents deploy the new hierarchy by reconfiguring the caches. The optimal hierarchy is based on the available bandwidth on the links between the caches and the inter-cache hit ratio. The system was validated by processing web traces on a Squid cache hierarchy. Performance improvement of up to 30% was observed with an adaptive hierarchy under varying network conditions.

Keywords: Web based systems, hierarchical web caching, distributed agents, adaptive system.

1 Introduction

The rapid growth of Internet has inspired researchers to develop new techniques to improve Web access performance. One such technique is Web Proxy Caching [1, 3, 11]. Advantages of caching include faster delivery of web objects to clients, reduced load on the servers, and lower Internet traffic. In order to further improve the performance of the Web and to cope with the growing demands of corporations, researchers have suggested networks of cooperating caches [4, 6, 8, 11]. This helps in distributing the load away from the server, and increases the probability of finding an object within the group of caches. A network of caches is commonly set up as a hierarchy in which a cache has a parent-child or a sibling-sibling relationship with other peer caches. A cache would first check if any of its peers have the needed document, else it would forward the request to a parent or obtain it from the web server. ICP (Internet Cache Protocol) is most commonly used for communication among caches [12], although many other protocols have been proposed. Deterioration in network conditions will slow down the exchange of information between the caches leading to a deterioration of performance in the cache hierarchy.

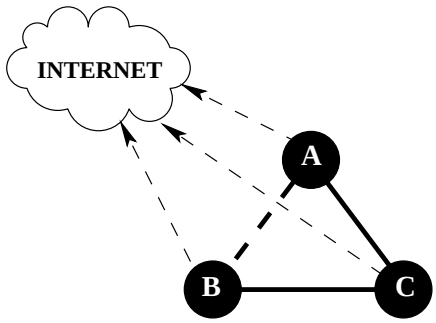

Figure 1. A simple caching hierarchy

Consider cooperating caches A, B, and C, that are siblings of each other in an ICP based hierarchy, as shown in Figure 1. Suppose the link A-B becomes congested, and as a result, data transfer between A and B slows down. The caches query their neighbors using ICP for the objects requested by their clients before retrieving them from the origin server. Hence, the retrieval of an object from the origin

server will take longer. Further, the exchange of objects between caches A and B will take longer, and it is conceivable that it may be faster to access an object from a server than a sibling cache. As a result, the clients of cache A and cache B will experience a higher latency.

If such a situation, which is not very rare, was to be avoided, the configurations of A and B must be changed so that they do not use each other as peers while the link A-B is busy. However, in current implementations, this will require manual reconfiguration of the caches. With fast changing network conditions, manually monitoring the network and reconfiguring the caches is not a feasible option.

In this paper, we address this problem by developing an Adaptive Hierarchy Management System that dynamically configures a set of distributed caches into hierarchies that are best suited to the existing conditions. The system is based on distributed agents that gather information, and a central controller that is responsible for finding and deploying a hierarchy among the caches. We evaluate the performance of an adaptive hierarchy designed by our framework by processing web traces on a network of Squid caches.

2 Discovering a good hierarchy

The goal of this research is to configure a given set of caches into the "best" hierarchy under existing conditions. For this paper, we restrict ourselves to all-sibling hierarchies. Hence, the goal is to find out for every pair of caches whether they should be siblings or not. We first analyze how a sibling helps the performance of a cache and then present a practical approach to automatically determining whether a cache should peer with another cache as a sibling.

2.1 Benefits of a web caching hierarchy

A web cache benefits from sibling caches in a hierarchy because some client requests are serviced by a sibling cache avoiding a trip to the origin web server. Sibling caches are checked before a request is sent to a web server, which unfortunately increases the service time for cache misses.

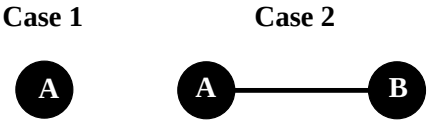

Figure 2. Cache A with and without a sibling cache B

We shall quantify the benefits a cache A gets by having another cache B as a sibling. The situation is illustrated in Figure 2. In Case 1, A sends all requests that result in a local miss to the origin server, while in Case 2, A first checks cache B, and sends only requests that are misses in caches A and B to the origin server. The metric used for comparison is the average response time for requests made by clients of cache A. The improvement in average response time observed by the clients of A due to a sibling can be approximated as follows:

$$Imp = ICHitratio * (servertime - siblingtime) - (missratio * ICPtime)$$

where:

- *Imp* = improvement in average response time going from an individual cache (Case 1) to a simple hierarchy (Case 2).
- *IChitratio* = Inter-Cache hit ratio. The fraction of requests by clients of A that are hits at B in Case 2.
- *missratio* = fraction of documents that have to be obtained from the origin web server in a hierarchy in Case 2.
- *servertime* = average response time for documents that have to be obtained from the origin web server, not including checking for documents in siblings (same for Case 1 and 2).
- *siblingtime* = average response time for documents that are obtained from a sibling cache (cache B in the example).
- *ICPtime* = average response time to check for documents in siblings.

We omit a derivation and detailed assumptions for this equation for brevity and simply state that it is representative of most realistic situations. In the equation, *servertime* is a constant. Hence, the main factors that determine the degree of benefit due to a hierarchy are *IChitratio*, *siblingtime*, and *ICPtime*.

The only way in which a hierarchy helps in improving performance is through hits in sibling caches. Hence, a relatively high inter-cache hit ratio (*IChitratio*) is critical for getting a tangible benefit from a hierarchy. The time to retrieve a document via a sibling cache (*siblingtime*) and the time to verify if a sibling cache contains a specific document (*ICPtime*) must be relatively low in order to get benefits from a hierarchy. Both of these strongly depend on network conditions. If the network connectivity is poor, it is entirely possible that slow communication between sibling caches can lead to a higher average response time when caches are in a cooperative hierarchy. The conclusion is that

inter-cache hit ratio and available network bandwidth are the key determinants of benefits that can be expected from a hierarchy.

2.2 Procedure to determine a good hierarchy

It is clear from the above discussion that the extent of performance benefit a cache can expect from a sibling primarily depends on the percentage of hits from the other cache and the available network bandwidth. We use a simple and practical procedure that uses threshold values of available network bandwidth and recent inter-cache hit ratio to determine if it is beneficial to have a sibling relationship between a pair of caches. The core of the procedure, say to determine if a cache B should continue to be a sibling of cache A, can be described as follows:

If (available-bandwidth $<$ *low-bw-threshold)*
  Remove B as a sibling of A
If (available-bandwidth $>$ *high-bw-threshold)*
  Keep B as a sibling of A
If (available-bandwidth $>$ *low-bw-threshold* && *available-bandwidth* $<$ *high-bw-threshold)*
  If (inter-cache-hit-ratio $>$ *hitratio-threshold)*
    Keep B as a sibling of A
  Else
    Remove B as a sibling of A

The thresholds in this procedure are determined empirically. The basic idea is as follows. Above a certain bandwidth (high-bw-threshold), it is practically free to access other caches, so a sibling relationship should be maintained. Below a certain bandwidth (low-bw-threshold), it is never profitable to contact a sibling, so the sibling relationship should be severed. If the bandwidth is in between, the sibling relationship should be maintained if the inter-cache hit ratio is high enough to outweigh the overhead of having a sibling.

The above procedure is simple, but practical and effective. The precise theoretical conditions for a sibling to be profitable that we discussed earlier can be difficult to compute. In our experience, the simplified procedure is just as effective and relatively easy to implement. The framework developed for hierarchy selection is designed so that any other algorithm for sibling selection can be plugged in easily. The thresholds in our procedure are determined by experimentation.

3 Adaptive hierarchy management framework

The goal of this research is to develop a system with the capability to dynamically configure a set of distributed caches into adaptive hierarchies without human intervention. We now describe our complete framework to achieve this goal. The framework consists of two major components: a centralized controller and distributed agents.

There is one agent per cache in the framework, which executes on the same machine as the cache. The main tasks of an agent are as follows:

- An agent collects the cache statistics and periodically reports them to the controller. For this research, the relevant information is the number of hits that the cache achieves from requests coming from each of the sibling caches.
- Agents are responsible for deploying a new cache hierarchy, although finding a new hierarchy is the task of the controller. Once an agent receives information about the global hierarchy, it reconfigures the local cache on its node to conform to that hierarchy. Hence, all agents combine to provide a mechanism for implementing a new hierarchy.

The controller has the following two principal tasks:

- The controller is responsible for collecting the available bandwidth information. The mechanism for monitoring bandwidth on the network is based on Network Weather Service (NWS) [14], a freely available bandwidth monitoring and prediction tool. The controller interacts with NWS agents to gather bandwidth information.
- The controller is responsible for determining the most suitable caching hierarchy by implementing the procedure discussed in Section 2. If it is determined that the caching hierarchy has to be changed, the controller broadcasts the information about the new hierarchy to the agents who then deploy the new hierarchy.

A setup of the adaptive hierarchy management framework with three caches is illustrated in Figure 3. The steps in the working of the adaptive hierarchy management system can be summarized as follows:

1. The controller retrieves available bandwidth information from NWS.
2. The controller requests the agents installed on the caches for inter-cache hit ratio information.
3. The agents gather the information from the caches and send it to the controller.
4. The controller designs the best hierarchy based on the hierarchy selection procedure. If the hierarchy is different from the existing one, it generates configuration details for each cache.

5. The controller broadcasts the configuration details of the new hierarchy to the agents.

6. The agents reconfigure the caches to conform to the new hierarchy.

7. Above steps are repeated periodically.

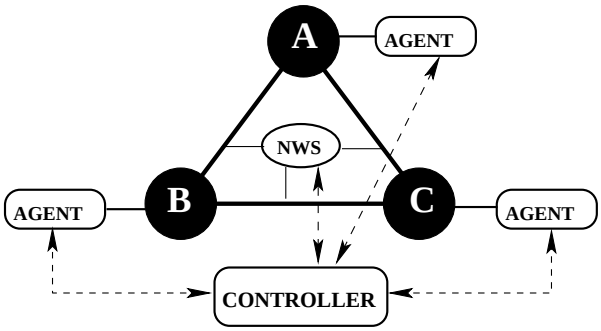

Figure 3. Adaptive Hierarchy Management System (Shows the interaction of the controller with the agents installed on the caches and with NWS)

4 Experiments and results

To evaluate the adaptive hierarchy management system, we compare the performance of a static all-sibling hierarchy to the case where the cache hierarchy is controlled dynamically by our adaptive hierarchy management system.

4.1 Experimental setup

We performed experiments on three Squid proxy caches. Two of the machines are 1.4Ghz Athlons with 512MB RAM running Redhat Linux 7.2. The third machine is an 800 MHz Pentium III with 512MB RAM running FreeBSD 4.3. All machines have 4GB reserved for Web caching and are installed with Squid 2.3.STABLE4.

The experiments are based on traces obtained from NLANR [9]. We randomly selected three daylong traces from different sites in the NLANR hierarchy. We selected approximately half million requests from each trace. We extract the URL field from the traces for use in the experiments. Simulated clients send the URLs to the caches at the rate of 20 requests/sec. The caches are directly connected to the Internet and serve the objects from their storage or from the web servers. To measure the performance, we use the mean and the median of the response time experienced by the clients of the cache for accessing the web objects. The results are filtered to remove very high latency values, typically caused by server error or non-existing documents. When needed, the available bandwidth on the links is controlled by the Dummynet toolkit [7].

4.2 Determination of threshold values

The procedure for determining a hierarchy depends on the thresholds for inter-cache hit ratios and available bandwidth. These thresholds are determined heuristically by running short sequences of traces in different available bandwidth and hit ratio scenarios. In our experiments, the available bandwidth is controlled by Dummynet and the hit ratio is controlled by adding more or less repetition in the trace. The procedure is done manually although it can be automated. We will not present the results to determine thresholds in detail but outline the types of experiments that were used.

Consider the two setups labeled Hierarchy0 and Hierarchy1 in Figure 4. The difference between the two setups is the presence or absence of a sibling relationship between cache A and cache B. We compare the performance of the caches in these two setups under varying bandwidth and inter-cache hit ratios. The results from cache A are presented in Figure 5. The results from cache B are similar, and hence omitted. As expected, the benefit from a sibling relationship is higher for higher hit ratios and lower when the bandwidth is reduced. We omit detailed results and discussion, but based on this suite of experiments, the thresholds were set as follows. The high and low network bandwidth thresholds were set to 10Mbps and 1 Mbps, respectively, and the inter-cache hit ratio threshold was set to 6%.

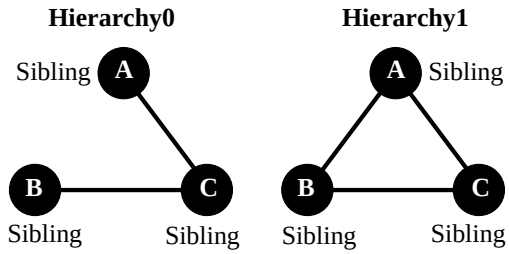

Figure 4. Different setups of an all-sibling hierarchy

4.3 Adaptive hierarchy performance

The performance of a three cache adaptive hierarchy was compared to a fixed hierarchy. During the experiment, the bandwidth between cache A and B varies randomly among phases of 100 Mbps, 10 Mbps, 1 Mbps, and 0.1 Mbps with equal probability. In the first experiment, all caches are each others siblings. In the second case, our adaptive hierarchy

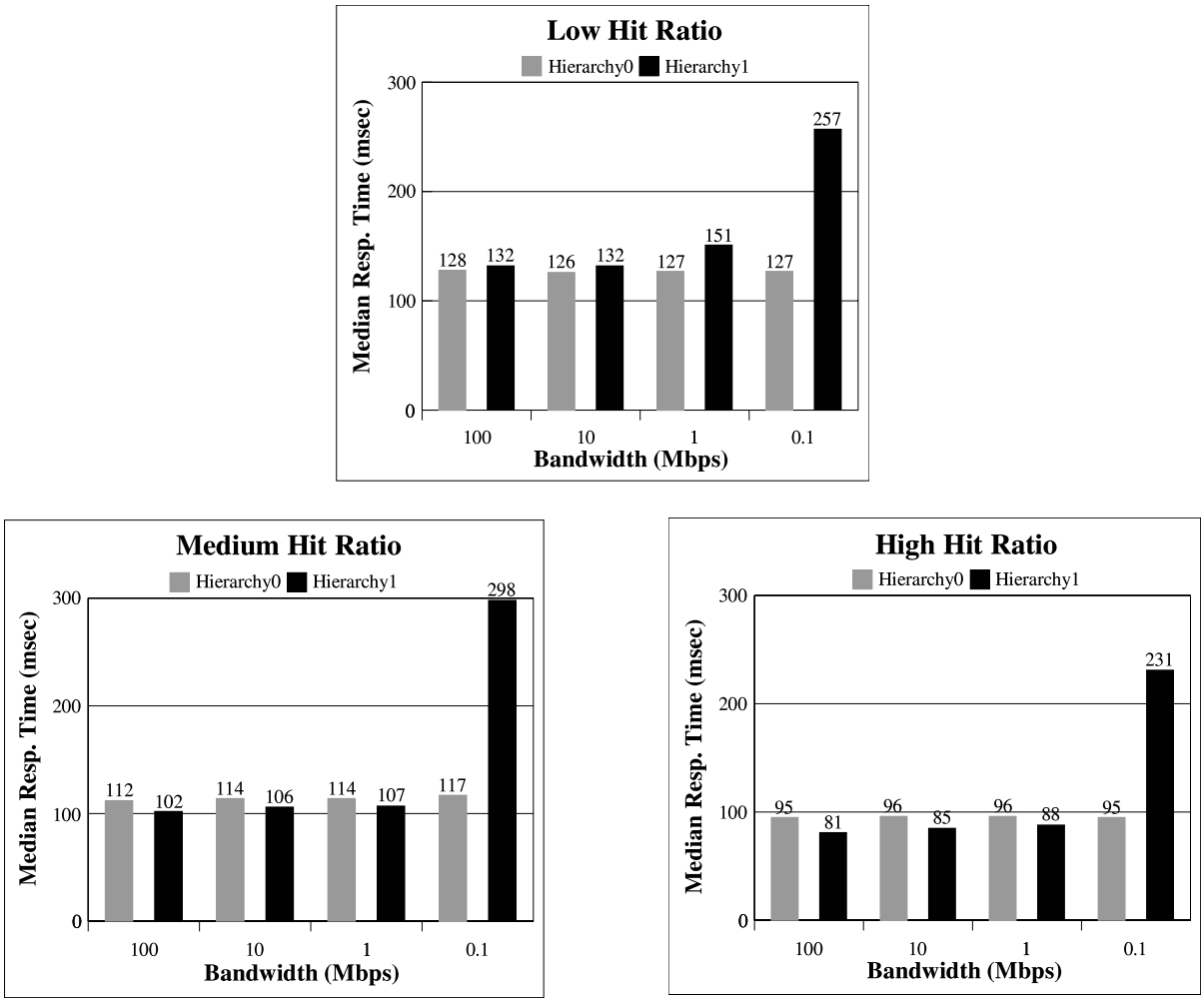

Figure 5. Median response times of clients of cache A under varying bandwidth on A-B link and different hit ratios

management system is used to determine and deploy the most suitable hierarchy periodically for the same trace. The results are shown in Figure 6 and Figure 7.

We observe from Figure 6 that the mean response times for cache A decreases from 0.439 seconds for a non-adaptive hierarchy to 0.388 seconds for the adaptive hierarchy, which is an improvement of 13%. For cache B, the performance improvement is about 29%. Median response time also improves but only by a small amount. The reason is that the adaptive hierarchy is mostly effective in low bandwidth conditions where the response times are high, and that does not impact the median since the data points still stay on the same side of the median.

Figure 7 shows the response times for individual requests. While the response time patterns with and without an adaptive hierarchy are similar for high bandwidth phases, it is clear that the response times for adaptive hierarchy are substantially shorter for low bandwidth phases. Response times with an adaptive hierarchy are fairly stable even in the face of fluctuating bandwidth, in sharp contrast to a fixed hierarchy.

5 Conclusion

This paper describes a system to automatically reconfigure a group of web caches into a hierarchy that is best suited to the existing conditions. Available bandwidth between the caches and inter-cache hit ratio are the criteria employed for building an appropriate hierarchy. The adaptive hierarchy management framework consists of distributed agents at the caches to collect information and deploy new hierar-

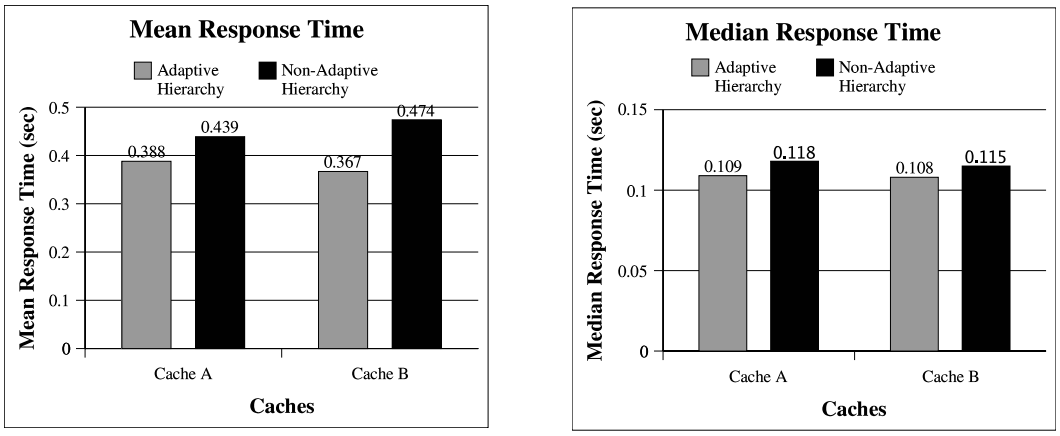

Figure 6. Performance of an adaptive hierarchy compared to a non-adaptive hierarchy

chies and a central controller that collects all information and creates the most suitable cache hierarchy.

The goal of this research is that a group of caches be able to take maximum advantage of each other, yet automatically adjust when the cost of cooperation exceeds the benefits. We have implemented the adaptive hierarchy framework developed in this paper and validated the benefits by running web traces on a testbed of three Squid caches. Results clearly demonstrate that an adaptive hierarchy performs significantly better and provides stable performance in varying network conditions.

6 Acknowledgments

This research was sponsored, in part, by the Texas Higher Education Coordinating Board under ATP program with grant number 003652-0424. Support was also provided by the Department of Energy through Los Alamos Computer Science Institute and by the Texas Learning and Computation Center. Compaq/HP loaned us the Polygraph testbed for this project. We thank Dr. Martin Herbordt at Boston University and Mr. Kevin Leigh at Hewlett Packard for their advise and help in this project.

References

[1] G. Barish and K. Obraczka. World wide web caching: Trends and techniques. *IEEE Communications Magazine Internet Technology Series*, May 2000.

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

[3] R. Cáceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich. Web proxy caching: The devil is in the details. In *Proceedings of the Workshop on Internet Server Performance*, December 1998.

[4] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell. A hierarchical internet object cache. In *Proceedings of the USENIX Technical Conference*, San Diego, CA, January 1996.

[5] B. Duska, D. Marwood, and M. Feeley. The measured access characteristics of world-wide-web client proxy caches. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS '97)*, December 1997.

[6] D. Povey and J. Harrison. A distributed internet cache. In *Proceedings of the 20th Australasian Computer Science Conference*, February 1997.

[7] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, Jan 1997.

[8] P. Rodriguez, C. Spanner, and E. W. Biersack. Web caching architectures: Hierarchical and distributed caching. In *The 4th International Web Caching Workshop*, San Diego, CA, April 1999.

[9] Traces (sanitized access.log). National Lab of Applied Network Research. Available at ftp://ircache.nlanr.net/Traces/.

[10] D. Wessels and K. Claffy. Evolution of nlanr cache hierarchy: Global configuration challenges, 1996.

[11] D. Wessels and K. Claffy. Application of Internet Cache Protocol (ICP), version 2, RFC 2187, July 1997.

[12] D. Wessels and K. Claffy. ICP and the squid web cache. *IEEE Journal on Selected Areas in Communication*, 16(3):345–357, April 1998.

[13] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *Proceedings of the 17th Symposium on Operating Systems Principles*, pages 16–31, December 1999.

[14] R. Wolski, N. Spring, and C. Peterson. Implementing a performance forecasting system for metacomputing: The Network Weather Service. In *Proceedings of Supercomputing '97*, San Jose, CA, Nov 1997.

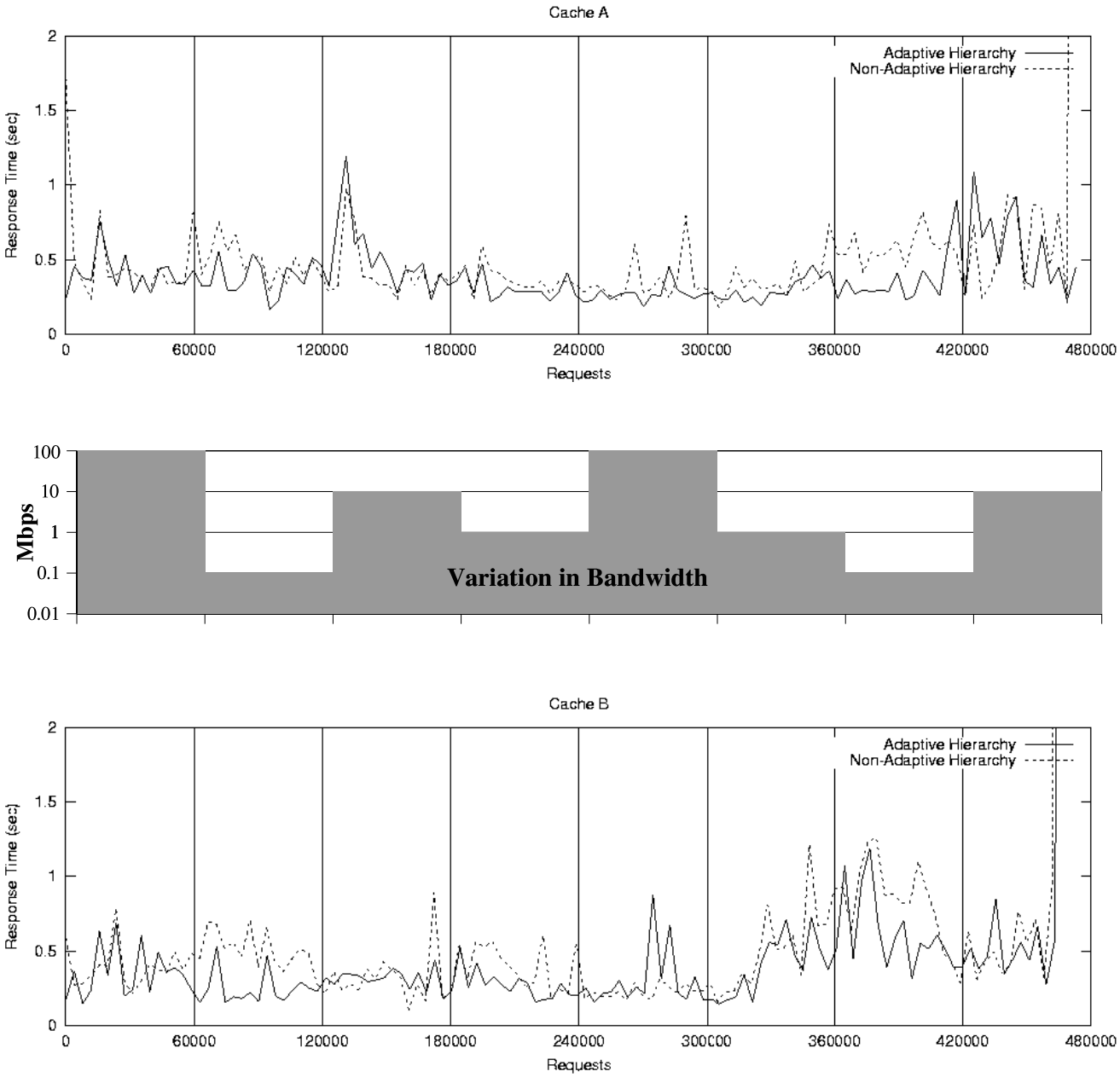

Figure 7. Response times for individual requests for an adaptive hierarchy and a non-adaptive hierarchy