ISSN:0975-9646

Performance Evaluation in Content Delivery Networks Using Fluid Queue Algorithm

Kothuru Srinivasulu , Nalini N

School of Computer Science and Engineering
VIT University, Vellore-632014, Tamilnadu, India

***Abstract*-Content Delivery Networks (CDNs) have evolved to overcome the inherent limitations of the Internet in terms of user perceived Quality of Service (QoS) when accessing Web content. This paper examines replication in content distribution networks and proposes a novel mechanism for optimally resolving performance versus cost tradeoffs. The key insight behind our work is to formally and analytically capture the relationship between performance, bandwidth overhead and storage requirements for a web cache, express the system goals as a mathematical optimization problem, and solve for the optimal extent of replication that achieves the desired system goals with minimal overhead. We describe the design and implementation of a new content distribution network based on this concept, called CobWeb. CobWeb can achieve a target lookup latency while minimizing network and storage overhead, minimize access time while keeping bandwidth usage below a set limit, and alleviate "flash crowd" effects by rapidly replicating popular objects through fast and highly adaptive replica management. We outline the architecture of the CobWeb system, describe its novel optimization algorithm for intelligent resource allocation, and compare, through simulations and a physical deployment on PlanetLab, CobWeb's informed, analysis-driven replication strategy to existing approaches based on passive caching and heuristics.**

***Keywords*:Contentdeliverynetworks,cobweb,control theory, Request balancing.**

I.INTRODUCTION

Demands made on their services. Such a scenario may cause unmanageable levels of traffic flow, resulting in many requests being lost. Replicating the same content or services over several mirrored Web servers strategically placed at various locations is a method commonly used by service providers to improve performance and scalability. The user is redirected to the nearest server and this approach helps to reduce network impact on the response time of the user requests Caching can significantly improve user perceived latencies as well as reduce the amount of aggregate network traffic. The popularity of the web makes caching a natural place to apply caching techniques to improve client performance, reduce server load, and minimize network traffic. Web caches to date have been deployed in two different settings, one driven by clients and one by content providers. Web caches that are placed close to the clients are commonly known as proxy caches. Such demand-side caches exploit temporal locality within the click stream of a single user as well as spatial locality stemming from the common interests of independent users. Proxy caches depend on passive monitoring and opportunistic caching, where each proxy only caches objects that have been requested by a client that is directly connected to it. Passive opportunistic caching severely limits potential benefits because web traffic is well-known to follow a Zipf distribution, with a heavy tail [3, 7, 1]. Since the heavy tail of the distribution limits spatial locality, past work has examined cooperative web caching, aimed at aggregating the cache contents of multiple web proxies to obtain greater caching benefits. Cooperative caching schemes that have been proposed include hierarchical , hash-based [1], directory based [9], and multi-cast schemes [2].

Yet past work on cooperative caching has examined only passive mechanisms for cache control, and an interesting negative result has demonstrated that cooperative caching provides performance benefits only within limited population bounds [3]. The large heavy-tail of the popularity distribution, combined with purely passive measures for cache control, makes it difficult to achieve high cache hit ratios. Web caches can also be placed within the network to aid content distribution. In particular, companies such as Akamai provide content distribution services to web site operators by placing servers in strategic locations to cache and replicate content. Such networks of servers are commonly known as content distribution networks (CDNs), and are driven by content providers rather than content consumers. In contrast to the demand-driven nature of web proxies, most CDNs proactively replicate web objects throughout the network using heuristics aimed at load balancing and improving performance [1, 9]. These heuristics aim to maximize the effective benefit from the bandwidth spent on proactive content distribution, but typically do not provide any hard performance guarantees. The fundamental challenge faced by any web cache is to decide which objects to replicate and to what extent. Proxy web caches sidestep this problem by passively caching objects that local clients have requested. In doing so, they limit the benefits that can be realized through caching to only those objects that have been fetched by the client population. CDNs, on the other hand, utilize heuristics which offer little control over the performance characteristics and resource consumption of the resulting system. For example, there is no way to guarantee a certain hit rate in such systems, or to cap bandwidth consumption at a desired limit. In this paper, we describe a novel, principled approach for determining which objects to cache and to what extent in a distributed CDN.We analytically model the costs and performance benefits of replication, formalize the tradeoffs as an

optimization problem, and use a novel numerical solver to find a near-optimal solution that maximizes
global system goals, such as achieving a targeted hit rate, while respecting resource limits, such as bandwidth consumption. Our system, CobWeb, is a global network of caching proxies that uses this analysis-driven approach, which utilizes the popularity, size, and update rate of web objects to compute the replication strategy. The resultant solution provides low latency lookup to clients while minimizing the storage and network overhead incurred by CobWeb proxies. Analytically modeling the overhead costs and performance benefits of replication enables CobWeb to convert this systems problem to an optimization problem. The optimization problem can then be solved to provide, for instance, minimal lookup latency while staying within a network bandwidth budget, or to achieve a targeted lookup performance while minimizing bandwidth consumption. This enables CobWeb to offer highly adjustable performance characteristics that is not available in heuristics based systems. Through simulations and measurements from a real world deployment, we make a case for structured, analysis-driven web caching over opportunistic heuristic driven caching. We show that our system provides high performance and low overhead when compared to passive caching systems, and propose deployment strategies for integrating our system into the Internet. The rest of this paper is structured as follows. In the next section, we describe the analysis-driven replication technique that enables CobWeb to resolve the performance-overhead tradeoff encountered in web caching. In Section 3, we outline the overall architecture of the CobWeb cache. Section 4 describes the current CobWeb implementation. In Section 5, we evaluate the performance of CobWeb through extensive simulations and a physical deployment on PlanetLab, and compare it to existing CDNs as well as passive caching.

II. Related Work

As a consequence, there has been an enormous growth in network traffic, driven by rapid acceptance of broadband access, along with increases in system complexity and content richness [7]. The over-evolving nature of the Internet brings new challenges in managing and delivering content to users. As an example, popular Web services often suffer congestion and bottleneck due to the large demands made on their services. A sudden spike in Web content requests may cause heavy workload on particular Web server(s), and as a result a hotspot [6] can be generated. Coping with such unexpected demand causes significant strain on a Web server. Eventually the Web servers are totally overwhelmed with the sudden increase in traffic, and the Web site holding the content becomes temporarily unavailable. The central insight behind CobWeb is that the fundamental tradeoff between performance and the cost required to achieve that performance can be treated as an optimization problem. CobWeb analytically models this tradeoff, poses it as an optimization problem, and finds the optimal replica placement strategy. This optimization analysis enables CobWeb to make informed decisions during replication in order to meet performance expectations with minimal cost. Conversely, this analysis can be used to optimize performance while keeping network and storage consumption below a fixed limit. CobWeb takes advantage of structured organization of the system to analytically model resource-performance tradeoffs. Several structured overlay systems, which organize the network to form well-defined topologies with regular node degree and bounded diameter, have been proposed in the recent past. These systems called Distributed Hash Tables (DHTs) provide high failure resilience and scalability through decentralization and self-organization. By layering CobWeb on a DHT we not only inherit its high failure resilience and scalability, but also leverage its regular topology to concisely capture performance-overhead tradeoffs. We illustrate this structured analysis using Pastry [9] as an example overlay. Pastry organizes the network as a ring by assigning identifiers to nodes from a circular identifier space. Objects are also assigned an identifier from the same space and stored at the node with the closest identifier, called the home node. When queries are injected into the system, Pastry routes the queries towards the home node by successively matching prefix digits in the identifier of the queried object. This routing process is aided by long distance contacts with different numbers of matching prefix digits and takes O(logN) hops in a network of N nodes. The structured organization provides an opportunity for replication to shorten the route of the lookup path. By replicating objects at all nodes that are within i hops from the home-node, the lookup latency can be reduced to log(N) □ i hops. We formalize this concept by defining a replication level for each object. An object is said to be replicated at level l if it is stored at all nodes in the system with l matching prefix digits. An l level object has lookup latency of l hops and is replicated at N bl nodes in the system. Figure 1 illustrates the concept of replication levels in Pastry. Structured replication of this manner enables CobWeb to concisely express the replication cost and lookup latency for each object. CobWeb extends this to analytically frame the global performance-overhead tradeoffs.

III. System Architecture

CobWeb operates as a globally distributed ring of cooperating nodes. Each CobWeb node acts as a Web proxy capable of serving any HTTP request. We envision that institutions that currently have large Web caches at their gateway to the Internet, will let the caches join the global CobWeb ring and share cache content intelligently and optimally. Other publicly available Web caches, such as Squid, may also be part of the CobWeb system taking the benefits independent users. The overall architecture of CobWeb is illustrated CobWeb distributes objects uniformly between its nodes through consistent-hashing [11]. Each web object is assigned a unique identifier that is a SHA-1 hash of its URL. When a CobWeb proxy receives a request from a client, it routes the request through the underlying overlay, directing the query toward the object’s home node, the node whose identifier is numerically closest to the object’s identifier. The first node along the routing path which has a copy of the object returns the object to the origin CobWeb proxy, which is responsible for delivering it to the client. Web objects are not loaded into CobWeb unless requested. When a URL is first requested, its home node is responsible for fetching the object from the origin web server and inserting it into the system. Subsequently, the home node is also responsible for renewing the object when it expires and propagating changes to other nodes. Non-cacheable web objects are simply delivered to the client but not stored within the CobWeb system. Home nodes also delete objects from the system if they do not receive any queries over a long period of time. CobWeb inherits high failure resilience from the overlay substrate. When a home node fails, the next closest node in the identifier automatically becomes the home node of an object. Objects for which home nodes has the sole copy, simply disappear from the system. This behavior is perfectly correct because CobWeb serves merely as a performance enhancing soft cache, rather than a permanent store. Moreover, popular objects would not be lost in this manner because they will be widely replicated in the neighborhood of the home node. Users access CobWeb in a transparent way without requiring any extensions or reconfigurations to the browser. In order take advantage of CobWeb, a user merely needs to append ".cobweb.org:8888" to the main URL of a web site. The http request is diverted to the closest CobWeb server through DNS-redirection. Subsequently, all web pages accessed through links on the main URL are automatically redirected through CobWeb. The latter is achieved through URL rewriting. Alternatively, CobWeb is also available as a conventional proxy service, which can be accessed by setting the proxy options in the browser to point to the closest CobWeb node. An important issue in any cooperative web cache is that a single compromised node can introduce misleading content into the system and launch phishing attacks. While, this is not a problem if CobWeb were to be deployed under centralized management, such as inside Akamai or on Planet-Lab, a collaborative environment poses security risks that need to be tackled. The security issue is further heightened because web objects are not self-certifying. To reduce this vulnerability, we propose a collaborative approach for certifying web content. A small quorum of CobWeb nodes can independently fetch objects and sign objects using a shared key exchanged through threshold cryptographic protocols [16, 15].

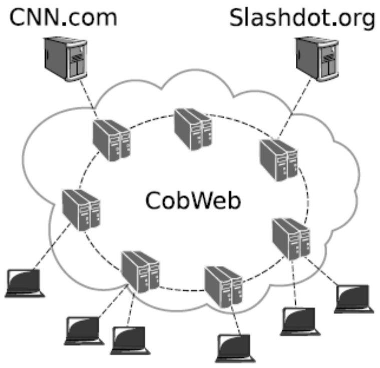

Fig1. System Architecture.

IV. Popularity Aggregation

The optimization algorithm described the performance and cost characteristics of the object. The object-specific cost information, such as the size, update rate, and server imposed load limit, can be stored and replicated along with the object. The workload-specific characteristics, that is, the query rate of an object, on the other hand, needs to be aggregated in the system, since queries are spread over all nodes caching that object. A naive way to compute the query rate of an object, is to have each node periodically measure, in some aggregation interval, the number of queries an object receives in a given period. However, if the query distribution is heavy tailed, as if often the case in web traffic [3], there can be several orders of magnitude of difference between the query rates of the popular and unpopular objects. Hence, no single aggregation interval would be large enough to accurately estimate the query rates of all objects and small enough to allow the system to detect rapid changes in the popularity of objects, which may arise during a flash crowd. One alternative is to measure the inter-arrival times of each object at each node and use those measurements to determine the query rate. However, since objects may be replicated at different nodes, any single node cannot estimate the global query inter-arrival time of an object. CobWeb uses a hybrid of the two approaches, namely query-rate estimation and inter-arrival time estimation. Nodes with cached objects measure the number of queries for those objects in each aggregation interval. Each node periodically transmits the data collected for each object towards the home node of that object. Each node along the path of the route aggregates the data they receive and continues to route the data toward the home node. Ultimately, each node receives a count of queries for all the objects for which it is the home node. To reduce aggregation overhead, CobWeb sends aggregation messages only if they are non-zero. This reduces the number of aggregation messages sent at each aggregation interval. Home nodes, then estimate the inter-arrival time using the aggregate query-rate received by it. For unpopular objects which may not be queried for in many aggregation intervals, the home node estimates the query inter-arrival time in terms of the number of aggregation intervals before a query is seen. That is, if an object receives one query every ith aggregation interval, it has a query inter arrival time of i. For popular objects, which many queries in the same aggregation intervals, it estimates their query inter-arrival time as 1=j, where j is the number of queries seen in a single aggregation interval. This technique allows us to choose very small values for the aggregation interval, which in turn enables CobWeb to quickly detect changes in the query rate and adapt accordingly.

Algorithm

```
//node status change
  Pro_space[0]=0;
Load-diff=0;
Load_dif_sum=0;
For(k=1;k<=n;k++){
 Fi(load_p –peer[k].load){
  Load_change= load_p –peer[k].load;
  New_pro_space(load_dif, Pro_space);
 Load_dif_change=load_dif_change+load_diff;
}
Change_pro_space(lod_dif_change,pro_space);
}
// load balanceing
 If(pro_space[]==NULL)
  Server_request();
Else
  Double y= rand();
Int req_send=0; int j=0;
While(pro_space[j]==1){
Send_to(peer[j-1].addr);
Req_send=1;
}j++}}
```

V. Implementation

The previous sections outlined the core distributed algorithms and mechanisms that enable a CDN to achieve high performance while respecting resource consumption constraints. In this section, we describe the CobWeb implementation and show how the algorithmic advantages of the analytical framework can be made practical, transparent and easy to use. CobWeb is implemented on Free Pastry v1.3, an open source implementation of Pastry [5]. Layering CobWeb on Pastry enables the system to build on the strong failureresilience, scalability, worst-case performance guarantees provided by Pastry, and to complement these properties with strong average-case performance guarantees. The CobWeb replication framework is practical and straight-forward to implement. Table 1 shows the size of the different components of the system. The total complexity of the numerical solver, combined with the high performance web cache front-end, is roughly comparable to the complexity of the Pastry overlay. In fact, most of the complexity resides in mundane issues like HTTP parsing, streaming content from multiple sources to clients, and coordination of concurrent threads, as opposed to the numerical solver. We envision that CobWeb will be deployed on server class hosts deployed close to the network core, under a single administrative authority. This is identical to the Akamai model as well as the current deployment model where our research group runs the open CobWeb cache on PlanetLab. Even though CobWeb is built on a peer-to peer proxy that can integrate any host anywhere, admitting poorly provisioned hosts located behind cable lines into the system is unlikely to offset the additional overhead they entail. Further, in a collaborative deployment, where nodes under different administrative domains are part of the CobWeb network, some nodes may be malicious and either attack the overlay or corrupt the content cached in the system. This problem can be easily solved if web servers provide digitally signed certificates along with content. An alternative solution that does not require changes to servers is to use threshold-cryptography to generate a certificate for content [14, 15]. When new content is to be inserted into the ring, the object can be fetched and partially-signed by a quorum of ring members. If the quorum size exceeds a threshold, partial signatures may be combined into a single signature that attests that t out of n nodes in a wedge on the CobWeb ring agree on the content. Such a scheme can ensure that rogue nodes below a threshold level cannot corrupt the system with bad content and other measures [4] can protect the underlying substrate from malicious nodes. However, the design and implementation of such a threshold-cryptographic scheme for a non-collaborative environment is beyond the scope

table1. Complexity of code

Component	Lines of code
FreePastry	17,712
Numerical Solver	6,163
Web Cache and Proxy	7,798

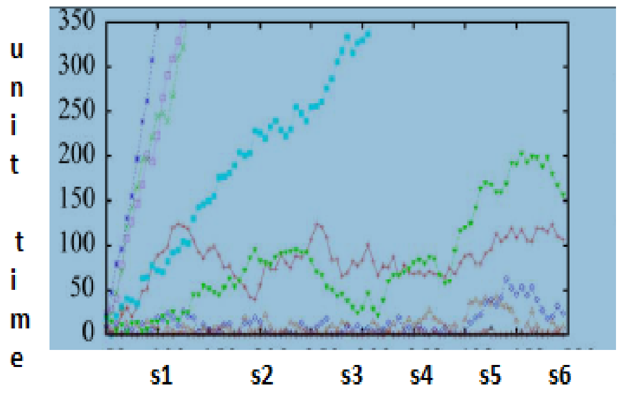

Fig2. Unit time & Server behavior

Conclusion

In this paper the fundamental tradeoff between performance and cost of web caches in an analytical model, and pose it as a mathematical optimization problem. We propose a novel algorithm and show that the optimization problem can be resolved in a near-optimal fashion. how our analytical model and its numerical solution can be implemented in a distributed fashion on a peer-to-peer substrate. The resulting content distribution network, CobWeb, benefits from the resilience and self-organizing properties of distributed hash tables, allowing it to scale and recover from failures. In addition, CobWeb is able to achieve a target lookup latency while minimizing network and storage overhead, optimize lookup latency while meeting a resource consumption budget, and adapt quickly to changes in workloads.

VI.References

[1] H. Ackermann, S. Fischer, M. Hoefer, and M. Sch¨ongens.Distributed algorithms for qos load balancing. In Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures, SPAA ’09, pages 197–203, 2009.

[2] C. P. J. Adolphs and P. Berenbrink. Improved bounds for discrete diffusive load balancing. In IPDPS, pages 820–826. IEEE Computer Society, 2012.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network Flows: Theory, Algorithms, and Applications. Prentice Hall, 1993.

[4] J. Allard, S. Cotin, F. Faure, P. Bensoussan, F. Poyer, C. Duriez, H. Delingette, L. Grisoni, et al. Sofa-an open source framework for

medical simulation. In Medicine Meets Virtual Reality, MMVR 15, 2007.

[5] D. Arora, A. Feldmann, G. Schaffrath, and S. Schmid. On the benefit of virtualization: Strategies for flexible server allocation. In Proceedings of USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE ’11), 2011.

[6] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: results and open problems. Parallel and Distributed Systems, IEEE Transactions on, 16(3):207–218, 2005.

[7] P . Berenbrink, M. Hoefer, and T. Sauerwald. Distributed selfish load balancing on networks. In Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA ’11, pages 1487–1497, 2011.

[8] D. P. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. Computational Optimization and Applications, 1:7–66, 1992.

[9] D. P. Bertsekas and J. N. Tsitsiklis. Parallel and distributed computation: numerical methods. Prentice-Hall, 1989.

[10] A. Caprara, H. Kellerer, and U. Pferschy. A PTAS for the multiple subset sum problem with different knapsack capacities. Information Processing Letters, 73(3-4), 2000.

[11] E. Chan-Tin and N. Hopper. Accurate and provably secure latency estimation with treeple. In NDSS. The Internet Society, 2011.

[12] A. Chawla, B. Reed, K. Juhnke, and G. Syed. Semantics of caching with spoca: a stateless, proportional, optimallyconsistent addressing algorithm. In USENIXATC, 2011.

[13] Y. Chen, R. H. Katz, and J. Kubiatowicz. Dynamic replica placement for scalable content delivery. In IPTPS, Proceedigs, pages 306–318, London, UK, 2002. Springer-Verlag.

[14] G. Christodoulou and E. Koutsoupias. The price of anarchy of finite congestion games. In STOC, Proceedings, pages 67–73, 2005.

[15] E. Cronin, S. Jamin, C. Jin, A. R. Kurc, D. Raz, Y. Shavitt, and S. Member. Constrained mirror placement on the internet. In JSAC, pages 31–40, 2002.

[16] M. Drozdowski and M. Lawenda. Scheduling multiple divisible loads in homogeneous star systems. Journal of Scheduling, 11(5):347–356, 2008.

[17] P. Dutot, L. Eyraud, G. Mouni´e, and D. Trystram. Bi-criteria algorithm for scheduling jobs on cluster platforms. In IPDPS, Proc., pages 125–132. ACM, 2004.

[18] M. Freedman. Experiences with coralcdn: A five-year operational view