Incentive-Based Propagation of Metadata Updates in Peer-to-Peer Networks

Mema Roussopoulos
Harvard University

Mary Baker
HP Laboratories

Abstract

As peer-to-peer networks become more popular, the use of metadata to achieve a variety of tasks (e.g. content location, pricing, auctioning) becomes more important. In this paper, we argue that propagating updates to cached metadata provides benefits beyond simple caching with a fixed expiration, yet requires application-specific incentive-based policies to moderate this propagation. We describe the issues involved in performing such propagation, provide a couple of examples where we have experimental evidence of the benefits of metadata update propagation, and propose a couple more examples where this propagation could be beneficial.

1 Introduction

As peer-to-peer networks gain more interest, the use of metadata to achieve certain tasks becomes more important. For example, one of the fundamental operations in current popular peer-to-peer networks is that of locating content: given the name or a set of keyword attributes of an object of interest, how do you locate the object within the peer-to-peer network? Many peer-to-peer networks push a set of metadata in response to a search query [gnu, RFH+01, SMK+01, RD01, ZKJ01]. This metadata typically consists of index entries that point to the locations of nodes that serve replicas of the content of interest, but could also include other information such as pricing, reputation or load information of each these serving nodes.

The peer-to-peer model assumes that the global set of valid metadata (index entries, pricing, etc.) will change constantly as peer nodes join and leave the network and content is continuously added to and deleted from the network. Nodes that use metadata to serve queries in a more timely fashion need to know about changes to the metadata to perform well. "Well" is

application-dependent, and could for example mean faster content download time or higher integrity of downloaded content. Keeping cached metadata up-to-date requires tracking which metadata items change, as well as tracking when interest in updating particular items at each cache has died down so as to avoid unnecessary update propagation.

In this paper, we argue that in a peer-to-peer network, propagation of updates to cached metadata provides significant benefits beyond simple caching with a fixed expiration, yet requires application-specific incentive-based policies to moderate this propagation. These policies cannot be assumed to be universally applied by all peers since each peer will have different capacity and/or motivation for participation in the peer-to-peer application.

Our work with CUP [RB03], a protocol for performing Controlled Update Propagation in a peer-to-peer network has allowed us to explore this possibility. CUP asynchronously builds caches of metadata while answering search queries. It then propagates updates of metadata to maintain these caches. A key feature of CUP is that it allows nodes to use their own incentive-based policies to determine when to receive and when to propagate updates.

We describe how CUP works and give a couple of examples where the use of CUP to propagate metadata updates has been very beneficial. Specifically, we first describe how CUP can be used to maintain caches of index entries, since this is the metadata many researchers have recently advocated caching (e.g. [RFH+01, SMK+01]). We show that by using propagation policies where the incentive to cut off propagation is *popularity-driven*, CUP can greatly reduce the average search query latency while recovering update propagation overhead. We then describe how we have leveraged CUP to deliver metadata required for effective load-balancing of content demand across replica nodes.

We also describe other kinds of metadata that could be propagated and other kinds of incentives that could drive the propagation decision including *capacity-driven* and *value-driven* incentives. Our main goal here is to present evidence that controlled propagation of metadata updates is crucial and to open discussion on new services we can achieve in a peer-to-peer network through effective metadata update propagation.

2 The CUP Protocol

We describe how CUP works using the example of maintaining caches of index entries. CUP is not tied to any particular search mechanism and

therefore can be applied in both networks that perform structured search as well as networks that perform unstructured search. In this paper, we will describe how CUP works for structured peer-to-peer networks. In structured search, queries follow a well-defined path from the querying node to an authority node that holds the index entries pertaining to the query [RFH+01, RD01, SMK+01, ZKJ01].

The basic idea of CUP is that every node in the peer-to-peer network maintains two logical channels per neighbor: a query channel and an update channel. The query channel is used to forward lookup queries for content of interest to the neighbor that is closest to the authority node holding the entries for that content. The update channel is used to forward query responses asynchronously to a neighbor. These query responses contain sets of index entries that point to nodes holding the content in question. The update channel is also used to update index entries that are cached at the neighbor.

Figure 1 shows a snapshot of CUP in progress in a network of seven nodes. The four logical channels are shown between each pair of nodes. The left half of each node shows the set of content items for which the node is the authority. The right half shows the set of content items for which the node has cached index entries as a result of handling lookup queries. For example, node A is the authority node for content $K3$ and nodes C,D,E,F, and G have cached index entries for content $K3$. The process of querying and updating index entries for a particular content K forms a CUP tree whose root is the authority node for content K. The branches of the tree are formed by the paths traveled by lookup queries from other nodes in the network. For example, in Figure 1, node A is the root of the CUP tree for $K3$ and branch {F,D,C,A} has grown as a result of a lookup query for $K3$ at node F.

It is the authority node A for content $K3$ which is guaranteed to know the location of all nodes, called *content replica nodes* that serve content $K3$. Replica nodes first send birth messages to authority A to indicate they are serving content $K3$. They may also send periodic refreshes or invalidation messages to A to indicate they are still serving or no longer serving the content. A then forwards on any birth, refresh or invalidation messages it receives, which are propagated down the CUP tree to all interested nodes in the network. For example, in Figure 1 any update messages for index entries associated with content $K3$ that arrive at A from replica nodes are forwarded down the $K3$ CUP tree to C at level 1, D and E at level 2, and F and G at level 3.

The cascaded propagation of updates from authority nodes down the

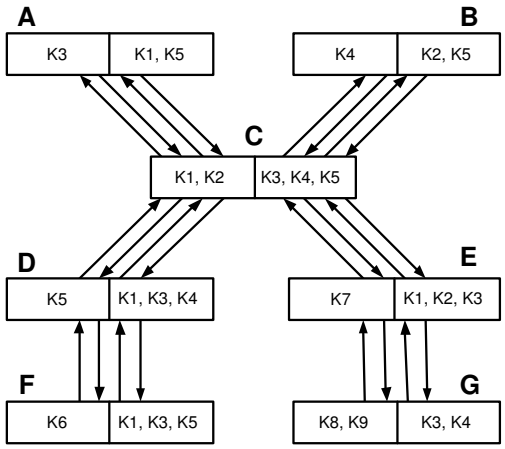

Figure 1: CUP Trees

reverse paths of search queries has many advantages. First, updates extend the lifetime of cached entries allowing intermediate nodes to continue serving queries from their caches without re-issuing new queries. It has been shown that up to fifty percent of content hits at web caches are "freshness misses", i.e., instances where the content is valid but stale and therefore cannot be used without first being re-validated [CK01]. Second, a node that proactively pushes updates to interested neighbors reduces its load of queries generated by those neighbors. Third, the further down an update gets pushed, the shorter the distance subsequent queries need to travel to reach a fresh cached answer. As a result, query response latency is reduced.

2.1 Popularity-Driven Incentives

In CUP, nodes are free to use their own incentive-based policies to determine when to propagate and when to receive updates. The incentive depends on the application goals and the metadata being updated. In the case of updating index entries, we have extensively studied policies using *popularity-based* incentives [RB03]. These are policies where the incentive to receive index updates for a particular content item depends on the frequency of queries for that item. The higher the frequency, the greater the incentive to receive updates for that item. Therefore, by pushing popular index updates to neighbors, the node saves itself from getting queries from those neighbors.

Popularity-based propagation policies aim to moderate the update overhead, i.e., the number of network hops traveled by updates. We briefly

describe the cost model we use when analyzing the performance of CUP under these policies, and we present some experimental evidence that such policies greatly reduce the average search query latency while at the same time recovering propagation overhead.

Consider a node A that is the authority for content K and consider the tree generated by issuing a query for K from every node in the peer-to-peer network. The resulting tree, rooted at A, is the spanning tree for K, S(A,K). This tree contains all possible query paths for K. We define cost in terms of network hops traveled. From a node N's perspective, the cost of a query for K that incurs a cache miss at N is two hops, one hop to push the query up to N's parent and one hop to push the answer down to N.[1] The cost of an update for K is one hop to receive the update from N's parent.

The propagation of an update for K to a node N is *justified* if its cost is recovered by a subsequent query arriving in the spanning subtree below N. For each hop a justified update u is pushed down to the root N of subtree S(N,K), exactly one hop is saved since without u's propagation, entries in all nodes of S(N,K) will expire simultaneously and the first subsequent query landing at a node N_i in S(N,K) within u's expiration will cause two hops, from N to its parent and back. This halves the number of hops traveled between N and its parent which in turn reduces query latency. In fact all subsequent queries posted somewhere in S(N,K) before u's expiration will benefit from N receiving u. Thus, the benefit of a justified CUP update goes beyond just recovery of its cost.

The cumulative benefit an update u brings to subtree S(N,K) increases when N is closer to the authority node since there is a higher probability that queries will be posted within S(N,K). We define "investment return" as the cumulative savings in hops achieved by pushing an update to N. As long as the investment return is greater than 1, CUP fully recovers its overhead.

To moderate propagation, we have explored two kinds of popularity-based policies: probabilistic and history-based. In the probabilistic policy, a node determines if a received update is justified by calculating the chances a query will arrive somewhere in its subtree to recover the cost of the update. In the history-based policy, a node determines if an update is justified by comparing the ratio of query arrival rate to update arrival rate in a sliding window of the last n update arrivals. If the ratio of queries to updates is greater than some threshold, the update is deemed justified. In either of

[1] We assume query responses flow down and are cached along the reverse query path. The cost model changes slightly if query responses flow directly from the authority node to the original querying node.

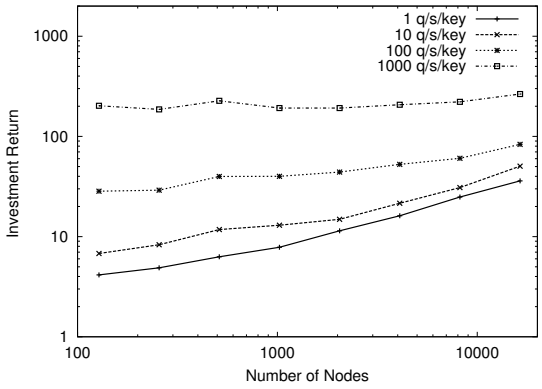

Figure 2: Investment return vs. net size. (Log-scale axes.)

the two policies, if the update is not justified, the node cuts off its intake of updates.

We have found that both types of popularity-based policies bring benefits. For example, when $n = 2$, the history-based policy can reduce search latency by as much as an order of magnitude when compared with path caching with fixed expiration [RB03]. Furthermore the update overhead this policy incurs is more than compensated for by its investment return. We see this in Figure 2 which shows the overall investment return for increasingly large networks and Poisson query arrival rates of 1, 10, 100, and 1000 queries per second. From the figure we see that for a particular network size, if we increase the query rate the investment return increases, and for a particular query rate, if we increase the network size, the investment return also increases. This demonstrates that CUP can scale to higher query rates and higher network sizes.

2.2 Load-Balancing Incentives

Having seen the benefits index updates bring, we have used the CUP protocol to study the problem of load balancing in a peer-to-peer network where the goal is to distribute the demand for a particular content fairly across the set of replica nodes that serve that content. In the literature, little focus has been given on how an individual node should choose among the set of index entries returned by a search query to forward client requests. This choice is important because peer nodes tend to be heterogeneous [SGG02] and some replica nodes will tend to have more capacity to satisfy requests for content than others. If nodes choose from the returned index entries

carefully, the system can serve content in a more timely fashion by directing content requests to more capable replica nodes.

Previous load-balancing techniques in the literature base their decisions on periodic or continuous updates containing information on *load* (e.g. [GC00, Mit97] or *available capacity* (e.g., [ZYZ+98]. We have leveraged the CUP protocol to deliver *load-balancing* metadata to see the efficacy of these algorithms when applied in a peer-to-peer context.

The incentive for a node N to receive a load-balancing update is driven by two factors: the popularity of the particular update at the subtree below N and how well the update contributes to balancing the demand for content across the set of replica nodes. By receiving a load-balancing update for a popular item, N not only benefits the replica nodes serving the content, but benefits itself as well. This is because without accurate load-balancing information, N might point clients to an overloaded replica. After failing to obtain the desired content from the replica, clients would then query N again for a different replica. N could thus save itself from getting such re-try queries if it receives the appropriate load-balancing updates.

When applied in a peer-to-peer context, each update propagated by previously proposed algorithms heavily depends on preceding updates propagated. For example, if some replicas report a high available capacity, this may cause peer nodes to forward an unpredictable number of requests to these replicas causing them to overload. The shift of the workload to these replicas increases the available capacities of other replicas who report higher availability at the next propagation causing requests to flock to them instead. This load oscillation can occur even when the workload request rate is as low as 60% of the total capacities of the replicas.

To avoid oscillation, we developed a new load-balancing algorithm, Max-Cap, that makes decisions based on the inherent maximum capacities of the replica nodes. We define maximum capacity as the maximum number of content requests per time unit that a replica claims it can handle. The maximum capacity is like a contract by which the replica agrees to abide. If the replica cannot sustain its advertised rate, then it may choose to advertise a new maximum capacity, which is propagated down the CUP tree. Max-Cap does not suffer from load oscillations as previous techniques do because the load-balancing updates it sends are independent of each other. Moreover, experiments show that Max-Cap incurs less propagation overhead than previous algorithms since load-balancing updates are only propagated when replicas choose to change their contracts or when replicas enter or leave the network [Rou02].

We cite our load-balancing work with CUP as more evidence that prop-

agation of metadata updates can be beneficial in that it provides the opportunity to achieve better application performance in a peer-to-peer network. In this case, better application performance means faster content transfer. However, this work is also an example of how controlled propagation of updates is necessary. Propagating all load-balancing updates as done by previously proposed algorithms based on available capacity does not guarantee good load-balancing performance in a peer-to-peer network. Max-Cap propagates load-balancing metadata updates in moderation, and it is this moderation we advocate here.

3 Other Incentives

3.1 Capacity-Driven Incentives

Nodes may not always be able to propagate all updates to all interested neighbors. When a node's outgoing capacity is limited, the incentive to propagate is *capacity-driven*. In this case, the cost model we introduced in Section 2.1 is extended to include the node's propagation capacity. For example, suppose a node N receives an update to be propagated to each of its child neighbors but N only has capacity to propagate to one of the children. The child, C, most justified in receiving this update is the one whose subtree S(C,K) will reap the largest investment return from receiving the update.

There are many other examples of possible capacity-driven propagation behavior. A node may favor pushing updates to neighbors that have higher connectivity over neighbors with lower connectivity. A node may favor neighbors that have been a good source of justified updates over neighbors that are less reliable or less cooperative. Or still a node may treat all neighbors equally but choose to re-order updates such that updates that provide higher benefits are pushed out first. For example, a node may push out older updates ahead of others that have more time left before they expire.

3.2 Metadata-Value-Driven Incentives

CUP can be leveraged to propagate other kinds of metadata as well. For example, the authority nodes could propagate prices advertised by replicas for services they offer. As the replica nodes change their offered price, the price updates could be propagated down the CUP trees to interested nodes. Replica prices may be a function of the replica's current connectivity, load,

or even the willingness of the replica to provide service. Clients then choose from amongst the replicas depending on the offered price advertised.[2]

Here, the incentive a node uses to cut off incoming price updates may be both popularity-driven as well as *metadata-value-driven*. Some kinds of metadata, such as price information may change frequently enough that receiving an update for every change is neither necessary nor desired. Whereas popularity-driven incentive policies mainly aim to reduce network overhead, value-driven incentive policies aim to reduce overhead incurred in processing and filtering unwanted or irrelevant fine-grained updates. Depending on the application, this overhead might consume computer resources (e.g., CPU, battery, etc.) but more importantly, may also consume human resources (if the filtering requires manual user intervention). The benefit of a justified update is thus not only reduced network hops traveled, but also reduced computer/human resources. The cost model of Section 2.1 must therefore take these into account as well.

Value-driven incentives to receive updates work much like current publish-subscribe applications, where subscribers indicate their interest in particular events or items (e.g., [RKCD01]). For example, a leaf node, N, may inform its parent that it would only like to receive price updates for replicas whose advertised price is less than some maximum value set by N's local clients. This way N's parent will not propagate updates for replicas whose service cannot be afforded by N's clients. Such a condition set at a leaf node would be propagated upward toward the authority root node. That is, a non-leaf node would cut off its intake of updates for replicas whose price exceeds the maximum price allowed by its local clients and the clients of nodes in the subtree below it.

Metadata update propagation can also enhance auctioning or bidding for service within a peer-to-peer network [CGM02]. Replicas that provide a particular service and receive bids for that service, can propagate these bids down the CUP tree. Nodes with clients interested in bidding for service join the CUP tree to receive these bid updates, allowing clients to set their bids accordingly. A variation of this bidding scenario is to have nodes with clients interested in bidding for service push updates to the authority node. The authority node then propagates these bid updates down the tree to serving nodes that are interested in providing the service.

As a final example, metadata update propagation can also enhance the integrity of content that is exchanged in a peer-to-peer network. Clients

[2]Connectivity and load information could be advertised by replicas and used together with price information in the client decision.

that receive service from replica nodes could report to authority nodes about the quality of the service received and this reputation information could be propagated down the CUP tree in effort to identify replica nodes that serve content cooperatively and with integrity.

4 Discussion

In this paper, we argue that propagation of metadata updates can be very beneficial in a peer-to-peer network. This propagation should be controlled, however, and should be guided by individual incentive-based policies. We propose CUP, a protocol for propagating metadata updates that allows peer nodes to use their own incentive-based policies to control update propagation. We present a couple of experimental examples of the benefits of moderated propagation as well as other potential applications that could take advantage of this. We hope this article opens up discussion concerning new services we can provide in a peer-to-peer network through effective propagation of metadata updates.

References

[CGM02] B. F. Cooper and H. Garcia-Molina. Bidding for storage space in a peer-to-peer data preservation system. In *ICDCS*, 2002.

[CK01] E. Cohen and H. Kaplan. Refreshment Policies for Web Content Caches. In *INFOCOM*, 2001.

[GC00] Z. Genova and K. J. Christensen. Challenges in URL Switching for Implementing Globally Distributed Web Sites. In *Workshop on Scalable Web Services*, 2000.

[gnu] The Gnutella Protocol Specification v0.4. http://gnutella.wego.com/.

[Mit97] M. Mitzenmacher. How Useful is Old Information? In *PODC*, 1997.

[RB03] Mema Roussopoulos and Mary Baker. CUP: Controlled Update Propagation in Peer to Peer Networks. In *Proceedings of the 2003 USENIX Annual Technical Conference*, June 2003.

[RD01] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *MiddleWare*, November 2001.

[RFH+01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *SIGCOMM*, 2001.

[RKCD01] A. Rowstron, A. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *NGC*, 2001.

[Rou02] M. Roussopoulos. *Controlled Update Propagation in Peer-to-Peer Networks*. PhD thesis, Stanford Univ., 2002.

[SGG02] S. Saroiu, P. K. Gummadi, and S.D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *MMCN*, 2002.

[SMK+01] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[ZKJ01] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, U. C. Berkeley, April 2001.

[ZYZ+98] H. Zhu, T. Yang, Q. Zheng, D. Watson, O. H. Ibarra, and T. Smith. Adaptive load sharing for clustered digital library services. In *HPDC*, 1998.