Backing up your DNS Cache

Xuebiao YUCHI, Likun ZHANG, Liming WANG, Anlei HU, and Xiaodong LEE

China Internet Network Information Center
Computer Network Information Center
Chinese Academy of Sciences, 100190 Beijing, China
{yuchixuebiao, zhanglikun, wangliming, huanlei, lee}@cnnic.cn

***Abstract*—Traditional ways to implement Domain Name System (DNS) cache backup are to dump the full DNS cache to some document/database periodically, which can bring great negative impact on DNS cache's performance. In this paper, based on a newly redesigned DNS cache architecture, we suggest an optimized DNS cache backup method in which a dynamic incremental DNS cache synchronization mechanism is introduced.**

Keywords-DNS; cache; backup

I. Introduction

The Domain Name System (DNS) is a distributed database that provides a directory service to translate domain names to Internet Protocol (IP) addresses [1] [2]. As one of the largest distributed system in the world, DNS consists of a hierarchy of name servers, with 13 root name servers at the top. For such a hierarchical system, caching is critical to its performance and scalability. To determine the IP address of a domain name, the DNS resolver residing at a client first sends a recursive query to its local DNS name server. If no valid cached mapping exists, then the local DNS name server will resolve the query by iteratively communicating with a root name server, a Top-Level Domain (TLD) name server, and a series of authoritative DNS name servers. All the replied DNS messages including referrals and answers are cached at the local DNS name server so that subsequent queries for the same domain name will be answered directly from the cache. Therefore, DNS caching significantly reduces the workload of root and TLD name servers, lookup latencies, and DNS traffic over the Internet.

Actually, caching has been an integral part of the DNS since its description in related RFCs. To support a certain degree of consistency, each DNS resource record has a Time-to-Live (TTL) value set by the administrator of the domain name specifying when it expires from the local DNS server's cache. Therefore, a cache miss will happen in the case of TTL expiration or first access to a domain name. In this case, the record must be re-fetched from the original authoritative DNS name server on query.

There have been many valuable studies conducted on performance of the TTL-based DNS caching mechanism during the past decade, in which the effect of varying TTLs on DNS cache hit rates are well explored [3] [4]. Note that besides the TTL factor, DNS cache miss can also be caused by first access to a domain name as mentioned above. Let us imagine the worst case: if all existing cached data are erased from DNS cache because of the local DNS name server's reboot for some unavoidable reasons (such as server crash), then the local DNS name server will suffer tremendous cache misses at the beginning of the reboot. Obviously, this result is not expected by operators of either local DNS name servers or authoritative DNS name servers, especially for these large Internet Service Providers.

An effective way to avoid this happening is to backup the DNS cache. Traditional means is to dump the DNS cache to some document/database periodically. Once the DNS cache is cracked down, the latest version of theses backup documents/databases can be loaded immediately after the DNS cache is restarted. In this way, a relative high cache hit rate can be well maintained. However, there are also some inevitable problems with this idea: first, a full copy of DNS cache needs to be read/written and transported during the dump/load procedure; second, the DNS cache must be locked during the dump/load procedure which can bring visible negative impact on DNS cache's performance. Obviously, both of them are not efficient for local DNS name servers, especially for these busy ones. To overcome these shortcomings, it is necessary to make some improvements on the DNS backup procedure. In this paper, based on a newly redesign of DNS cache architecture, we suggest an optimized DNS cache synchronization mechanism between two DNS caches, in which the DNS cache can be backed up to the other one dynamicly and incrementally and most of the important, the DNS cache's performance can be well maintained during this procedure.

The rest of this paper is organized as follows: Section II introduces our DNS cache requirements in which the overall DNS cache architecture is redesigned here. Section III details our proposed DNS cache backup mechanism. Finally, we conclude with a discussion of our work in Section IV.

II. DNS Cache Requirements

When investigating DNS query statistics of some large local DNS name servers, we find that only a small part of domain names are queried frequently [5], therefore, if the

This work is supported by the China Next Generation Internet Project (Project name: the Next Generation Internet's Trusted Domain Name System; No.: CNGI-09-03-04).

information of these domain names are cached properly, the performance of the DNS cache will be well improved.

Note that the basic unit of DNS cache should be DNS message. Each DNS message can be stored in DNS cache for a period of time determined by the TTL value of resource record in it. Since DNS messages in DNS cache may share common resource record set (RRset), to reduce the memory usage of DNS cache, it is better to distribute DNS message into two separate caches: Message Cache and RRset Cache. Furthermore, because most of the DNS messages we observed belong to class 'IN', in order to save the 16-bit class processing time when querying in DNS cache, a class-specific Message/RRset Cache is considered here. Therefore, DNS cache holds a list of class-specific Message/RRset Caches. When getting a DNS message, dispatch it to a proper Message/RRset Cache based on class. A typical Message (RRset) Cache manages a list of Message (RRset) entries which are held in a hash table. As illustrated in Fig. 1, entries in Message Cache only need to include some index information for RRset entries.

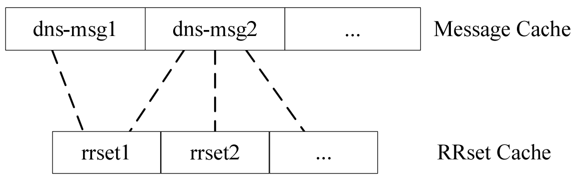

Fig. 1. Different entries in Message Cache may share the same RRset entry. On the other hand, one Message entry may contain multiple RRset entries' index information.

The overall relationship between these objects mentioned above is given in Fig. 2.

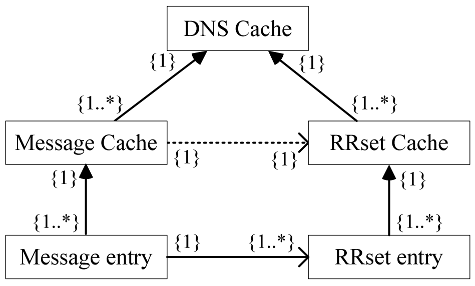

Fig. 2. Overall relationship between these different objects. Note that one Message entry could include index information for multiple RRset entries.

In the rest of this section, the broad requirements of the Message/RRset Cache will be detailed respectively.

A. RRset Cache

The key for one RRset entry in RRset Cache should be "rrset_name + rrset_type". In the value part, besides the RDATA of each resource record in the RRset, there should be the signature of RRset (RRSIG record) if it has, the authoritative level (for the ranking, see section 5.4.1 in [6]), and the security status (DNSSEC validation result) of RRset.

The expiration time for RRset entries in RRset Cache should be "now + TTL", any expired RRset should never be used again. Once the local DNS name server gets a new query result, the RRsets in the query result should be used to update the RRset Cache if the RRsets are more authoritative than those in RRset Cache.

In order to support dynamic DNS cache backup, the cached data should can be dumped out, loaded from one file, and edited in runtime. For RRset Cache, some key functions are required:

Looking up RRset

It must be possible to look up RRset entry in the RRset Cache: if the entry exists, return the information of this entry, or else return NULL.

Adding RRset

If the added RRset does not exist in RRset Cache, this RRset should be inserted as a new entry, or else update the existing RRset entry according to the new value.

Deleting RRset

The interface for removing one RRset entry from the RRset Cache should be provided, since it may be used by some third-part tools which can edit the RRset Cache directly. The simple way to delete one entry is to mark it as expired and to re-fetch it.

Updating RRset

Since some RRset entries may be updated before their expiration, the interface for updating RRset should be provided. When a non-expired RRset entry is updated with a new one, the new one should be more authoritative than the existed entry.

Dumping/loading to/from one document/database

The content of RRset Cache can be dumped to one document/database, so that the RRset Cache can be reused when the local DNS name server is restarted. Extra RRset expire time should be added when dumping, so that expired RRsets can be ignored when loading RRset Cache from the document/database.

B. Message Cache

The key for one Message entry in Message Cache should be "query_name + query_type" and the value part should include message header, index information for each RRset in different sections, see the following sketch in Fig. 3. In addition, the security status (DNSSEC validation result) and the authoritative level (authoritative or non-authoritative) of the Message should also be cached.

The expiration time for Message entries should be "now + TTL (the lowest TTL of RRsets associated with them)". Once a Message entry expires, it should be re-fetched when it is used. The DNS cache user should send out the query to some authoritative DNS name server and update the Message Cache when get the query result. Meanwhile, the RRset Cache should be updated together with Message Cache. Note that when updating the RRset Cache, whether to accept the

RRset in the query result, or retain the RRset already in the RRset Cache, the relative authoritative level of the data should be considered.

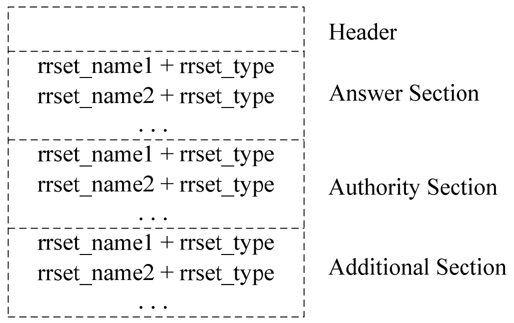

Fig. 3. The structure of a typical Message entry. Note that it does not include any index information for RRSIG records, since they are saved together with the original RRset as one of its attributes.

Similar to RRset Cache, some key functions required are list below:

Looking up Message

It must be possible to look up Message entry in the Message Cache:

If the entry expires or does not exist, return NULL.

If the entry exists and does not expire, then lookup its corresponding RRsets in the RRset Cache. If one RRset expires or does not exist in the RRset Cache, return NULL, or else, generate one real reply DNS message and return it.

Adding Message

The interface for adding Message entries to the Message Cache should be provided. When one Message entry is added, its corresponding RRsets should be used to update the RRset Cache, or insert into the Cache if they are not existed.

Deleting Message

The interface for removing Message entry from the Cache should be provided. The simple way to delete one entry is to mark it as expired and to re-fetch it.

Updating Message

Authoritative Message should never be updated by non-authoritative Message.

Dumping/loading to/from document/database

The content of Message Cache can be dumped to one document/database, so that the Message Cache can be reused when the local DNS name server is restarted. Extra Message expire time should be added when dumping, so that expired Messages can be ignored when loading Message Cache from the document/database. Note: RRset Cache should be loaded first.

C. Other Considerations

1) In order to gain higher efficiency, these Caches should be size-limited and configurable even in runtime-mode either bigger or smaller. To determine which entry should be removed when the Cache size reaches its limitation, each Cache keeps a LRU (Least Recently Used) list to track the managed entries by holding a list of pointers to them. In this case, when an entry is created or accessed, it should be moved to the end of the LRU list. Once the Cache is resized smaller or reaches its size, the entries at the head of the list should be removed.
2) The RRset/Message Cache must be thread-safe and it will be useful in debugging if a way to completely empty the Cache is provided.
3) The local DNS name server should provide the service during the DNS cache loading procedure if it takes too long time. (Hint for the implementation: If a short time, we can construct the DNS cache before we begin operations; if a long time, a separate component that works through the procedure in the background and does an “add if not here and if the stored data has not yet expired and is more authoritative” should work.)
4) The supported class should be configurable. Therefore, the local DNS name server should reject the query with the class not configured. [1]

Based on these requirements described above, a typical DNS cache query algorithm can be generalized as follows:

1) When the DNS cache gets a query, finds the Message Cache of the query class. If it cannot be found, return NOT_FOUND error.
2) Find the Message entry in the Message Cache according the key “query_name + query_type”. If it cannot be found, return error NOT_FOUND error.
3) Check whether the Message entry has expired. If it expires, return MESSAGE_EXPIRED error, or else go to next step.
4) Find RRset entries in the class-specific RRset Cache according to the RRsets information recorded in Message entry. If any RRset has expired, return RRSET_EXPIRED error, or else go to next step.
5) Generate the replied message according to the Message/RRset entry information. (Note: If the query does not request DNSSEC records, all the RRSIG records, if they were cached, should be removed from the generated message.)

III. DNS Cache Backup

Based on these DNS cache requirements described in the above section, our proposed DNS cache backup mechanism will be detailed here.

A. Basic Idea

Data in DNS cache is transient, which requires that the backup procedure should be implemented dynamically (or periodically). Furthermore, the backup should also be

[1]Since class CH is useful for getting ID.SERVER and the like but nothing useful in the caching side, and the only other class is HS which MIT can supply a patch for, it is also recommended that only the class IN needs to be supported by the DNS cache implementation.

incremental in order to achieve high performance and save the bandwidth. Therefore, the DNS cache being backed up needs some kinds of updating lists to record changes in it during a specific backup cycle: one deleting list to record indexes for these entries to be deleted and one adding list to record indexes for these entries to be added. Suppose we need to back up DNS cache *A* to DNS cache *B* as shown in Fig. 4, therefore, all Message/RRset Caches in *A* need to keep their own lists of these two kinds, respectively. Then, based on these lists, the DNS cache *B* can be updated incrementally.

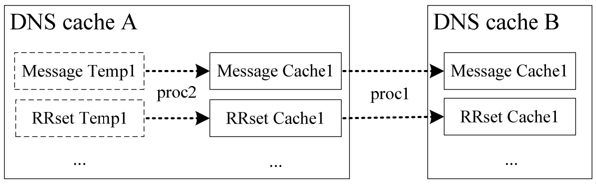

Fig. 4. A typical DNS cache backup cycle from DNS cache A to DNS cache B.

Note that these updating lists should be kept unchangeable when they are utilized during DNS cache *B*'s updating procedure (namely *proc1* in Fig. 4), so their corresponding Message/RRset Caches should not be updated during *proc1*, in other words, these Caches should be kept read-only. Therefore, some temporary Caches are needed to store these new arrival entries during *proc1*.

When *proc1* is completed, these updating lists will be cleared and the read-only lock on Message/RRset Caches in *A* will be removed. Then in procedure *proc2*, these Temp Caches will be locked in which these new arrival entries will be merged into their corresponding Message/RRset Caches in *A* (Hint for the implementation: add if not here and if the merged data has not yet expired and is more authoritative).

After performing procedures *proc1* and *proc2*, a complete backup cycle is finished.

B. *Other Considerations*

1) The interval between two backup cycles can be configurable, being a trade-off between performance and consistency. Furthermore, for performance consideration, these updating lists should also be size-limited. In this case, once one of these updating lists reaches its limitation, next backup cycle should be triggered immediately even though the configured interval has not yet arrived.
2) At the beginning of the backup, the initial adding lists should be the same as the LRU lists on DNS cache *A*, while the initial deleting lists should be empty. Note that this also applies with the case that the DNS cache *B* is restarted during the backup.
3) When performing procedure *proc1*, if the total size of adding list and deleting list for a specific Message/RRset Cache exceeds the Cache's current size, it is better to dump the overall Cache directly in order to improve performance.

IV. DISCUSSIONS

Since these updating lists kept in DNS cache *A* are configured to be size-limited, the procedure *proc1* can be carried out very quickly, therefore, the size of these Temp Caches generated during this period cannot be too large (DNS cache's average cache miss rate can be lower than 10% [3]), so the merging procedure *proc2* can also be carried out quickly within a few microseconds which can do little negative impact on the local DNS name server's performance.

As the key component of local DNS name server, DNS cache's performance is critical to the local DNS name server's service. During our DNS cache backup procedure, only some Temp Caches and updating lists are needed, which could do little negative impact on the DNS cache's overall performance. As of this paper is completed, the redesigned DNS cache structure has been applied to the BIND10 project [7] and the DNS backup mechanism introduced here will also be deployed in the near future even though it is only some initial thoughts so far. Obviously, these ideas still need to be tested through long-term practice to get further improvements.

ACKNOWLEDGMENT

We are grateful to Cindy Wang and Shane Kerr in Internet Systems Consortium (ISC) for several useful advices especially in optimizing the redesign of DNS cache architecture.

REFERENCES

[1] P. Mockapetris, “Domain names - concepts and facilities,” RFC 1034, Nov. 1987.
[2] P. Mockapetris, “Domain names - implementation and specification,” RFC 1035, Nov. 1987.
[3] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, “DNS performance and the effectiveness of caching,” *IEEE/ACM Transactions on Networking,* 10(5), 589—603, 2002.
[4] J. Jung, A.W. Berger, and H. Balakrishnan, “Modeling TTL-based Internet caches,” *Proc. IEEE INFOCOM '03*, Apr. 2003.
[5] X. B. Yuchi, X. Wang, X. D. Lee, and B. P. Yan, “A new statistical approach to DNS traffic anomaly detection.” *In Proc. 6th International Conference on Advanced Data Mining and Applications (ADMA2010),* LNCS, vol. 6441, pp. 302-313, Springer, Heidelberg, 2010.
[6] R. Elz, R. Bush, “Clarifications to the DNS specification”, RFC 2181, Jul. 1997.
[7] ISC BIND 10 project, http://www.isc.org/bind10/project/.