DoCR: DNS with Strong Consistency for Update-frequent Environment

Yuhan Zhou
zhouyuhan@pku.edu.cn
Peking University
Beijing, China

Wenrui Liu
liuwenrui@pku.edu.cn
Peking University
Beijing, China

Chenren Xu*
chenren@pku.edu.cn
Peking University
Beijing, China

ABSTRACT

Domain Name System (DNS) is a critical infrastructure for the Internet. Now deployed DNS servers use Time To Live mechanism to update name-IP mappings. Although this mechanism only reaches weak consistency, it suffices since DNS records are rarely updated. However in an IP update-frequent environment this TTL method would be problematic. This paper describes the design, implementation and evaluation of *DoCR*, a new DNS service framework that reaches strong consistency and is compatible with now-existing DNS infrastructure. We show that *DoCR* adds almost no read overhead, acceptable write overhead and its ability to tolerant node crashes. *DoCR* could be deployed in a name zone where lots of mobile servers are using domain name service.

CCS CONCEPTS

• **Networks** → **Naming and addressing**.

KEYWORDS

DNS, CRAQ, strong consistency

ACM Reference Format:
Yuhan Zhou, Wenrui Liu, and Chenren Xu. 2021. DoCR: DNS with Strong Consistency for Update-frequent Environment. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The Domain Name System (DNS) is a distributed database which provides a directory service to translate domain names to IP addresses. In this kind of hierarchical system, caching is an important mechanism for reducing workloads and latency, but caching incurs inconsistency between cached copies and reference records. Such cache consistency problem could be a serious concern in an environment where IP address update is frequent. To reduce inconsistency, current DNS servers employ Time-To-Live(TTL) value for each record. The server deletes the corresponding record when the TTL value decreases to zero.

Unfortunately, the traditional TTL mechanism only supports weak consistency. The client may still get an outdated copy from the cache. Strong consistency, which is defined as providing the guarantee that a read to a record always sees the latest written value, cannot be achieved. This weak consistency does not incur great concerns because name-IP mappings are rarely changed nowadays.

However, DNS consistency becomes a great problem in an IP update-frequent environment. With the popularization of mobile devices, mobile phones can now acting as Linux servers with the help of some plug-in applications(e.g. Busy Box, Linux Deploy). While cellular network can provide a constant IP to a mobile server, WiFi would be a better choice since it reduces cost. If a mobile server uses both cellular and WiFi network or it roams around different APs, its IP would change frequently and clients may not reach it by acquiring a stale record from DNS servers.

There has been some approaches to address this problem and basically they can be divided into two classes: (1) Adapt a dynamic TTL based on the update or query frequency of a record. Such approaches propose a metric for DNS update frequency(e.g. *EAI* in *ECO-DNS* [1]) and a TTL allocation scheme (e.g. *DNScup* [2]). These approaches make few modifications to exising DNS servers but their strong consistency are not fully proved. (2) Develop a novel Name Service System such as *DMAP*[5] and *Auspice* [3]. These approaches decouple the identifier and location information for a network host and reach a global consistent name service. However they require a new global infrastructure and now-existing DNS service has to be replaced, which is costly.

In this paper we propose *DoCR(DNS-on-CRAQ)*, a third approach that reaches strong consistency and is compatible with existing DNS mechanism. We use *CRAQ* [4], which is a method for replicating data across multiple servers to reach consistency and a DNS server behaves like a node in a *CRAQ* chain. The following sections present the design, implementation, and evaluation of our system.

2 DESIGN

DoCR consists of a *coordinator* acting as DHCP server and some chained *nodes* acting as DNS server replicas. The DHCP server handles all IP update requests from clients: it allocates an IP and deliver this update message to the head of the chain. The chain manages reading and writing on data in the following way.

- A node stores data as DNS items. A DNS item consists of multiple versions of a domain name, each includes a monotonically-increasing version number and an additional flag indicating whether the version is clean or dirty. Initally, all versions are marked as clean.
- When a node receives an update for a domain name, the node add this version to its list of the corresponding item. If the node is not the tail, it marks this version as dirty, and propagates the write to its successor. Otherwise, it marks

Conference'17, July 2017, Washington, DC, USA
© 2021 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

  the version as clean and notify all other nodes by sending an acknowledgement backwards.
- When an acknowledgment for an item version arrives at a node, the node marks the version as clean and delete all prior versions.
- When a node receives a query request for an item, if the latest known version of the requested item is clean, it returns this value. Otherwise, the node sends an ask message to the tail for the last clean version. The node then returns that version of the item.

The whole name-IP address mapping update procedure is illustrated in Figure 1. When a domain name owner requests to refresh a new IP address from DHCP server, the DHCP server sends a write message to the head node. After updating all DNS nodes, the head replies a message to the DHCP server, which then replies an update finished message to the domain name owner.

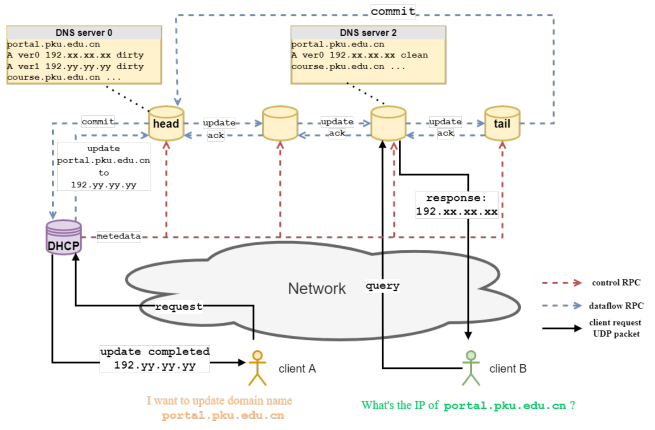
commit
DNS server 0
portal.pku.edu.cn
A ver0 192.xx.xx.xx dirty
A ver1 192.yy.yy.yy dirty
course.pku.edu.cn ...
DNS server 2
portal.pku.edu.cn
A ver0 192.xx.xx.xx clean
course.pku.edu.cn ...
commit
head
update
ack
update
ack
update
ack
tail
update portal.pku.edu.cn to 192.yy.yy.yy
response: 192.xx.xx.xx
DHCP
metedata
Network
query
request
update completed 192.yy.yy.yy
client A
client B
I want to update domain name portal.pku.edu.cn
What's the IP of portal.pku.edu.cn ?
control RPC
dataflow RPC
client request UDP packet

Figure 1: Name-IP mapping update procedure
Client A is trying to update the IP of a domain name. Before the DHCP server replies with a completed message, client B still sees the old mapping. After client A gets a comfirmation, all DNS servers have the new mapping.

3 IMPLEMENTATION

We build our system based on an open-source implementation of *CRAQ* (**https://github.com/despreston/go-craq**). Each *node* in *CRAQ* is adapted to a single DNS server. For simplicity, the *coordinator* is adapted to the DHCP server, which means it handles both IP update requests and node management. To compare the performance between our system and traditional TTL method, out system can be compiled in two modes.

3.1 Mode CRAQ

When compiled with `MODE=CRAQ`, our system performs what we describes in §2. The DHCP server must be activated first and it delivers metadata to the following activated DNS servers to let them know their positions in the chain. All communications between any two servers in the cluster (i.e. the DHCP server and the chain) use TCP-based RPC. Clients send their IP update requests to the DHCP server and domain name queries to any DNS servers in the chain in UDP packets, whose formats are fully described in RFC1035(**https://datatracker.ietf.org/doc/html/rfc1035**).

3.2 Mode TTL

When compiled with `MODE=TTL`, our system performs traditional TTL method on IP changes. The DHCP server only handles IP update requests and delivers them to the *primary* DNS server, which is the first activated server in all DNS servers. Other DNS servers get a name-IP mapping from the *primary* and caches it until its TTL expires.

4 EVALUATION

We try to compare the performance of DoCR and traditional TTL mechanism. At a high level, we are interested in read latency based on accurate queries, hit rate and overhead.

4.1 Experimental Setup

Testbad. We use a single workstation described in table 1 as our test machine. Without explicit mention, we run one DHCP process and two types of number of DNS nodes: three and nine. We simulate DoCR with RPC-connected DNS nodes and TTL DNS servers with isolated nodes. UDP sockets are used to connect the writing/reading clients with the DHCP/DNS servers.

CPU	Intel Core i5-5200 2.2GHz × 4
GPU	NV118/Mesa Intel HD Graphics 5500
RAM	3.8GiB
Disk	516.1GB
OS	Ubuntu 20.04 LTS

Table 1: Test machine configuration

Targets. The tests contain two types of workloads: low and high updating frequency. The interval bewteen sending IP change requests is used to control the frequency. Respectively, We set the low frequency interval to 10 seconds, and the high frequency interval to 2 seconds. At the beginning, we initiate ten domain name mappings in the DNS servers. By default, we select three IP addresses in sequence to renew for each update. We focus on various types of test indicators: result accuracy, IP change request RTT and DNS query RTT. Moreover, we test the fault recovery of DoCR.

4.2 Performance

DNS query accuracy. Figure 2,3 compare the DNS query accuracy between DoCR and the traditional DNS server at low and high IP change frequency respectively. For each comparison, we send a DNS request to the header(*primary*) and an intermediate node. Because the header node always has the latest mapping, we can determine the correctness of DNS responses by comparing the two results. We set each mapping item's TTL to 15 seconds when testing the traditional DNS server. Both in the low and high IP change frequency, DoCR can achieve almost 100% accuracy, which means strong consistency among these DNS nodes. The traditional DNS server cannot response the correct DNS mapping until TTL expires, which means weak consistency among these DNS nodes.

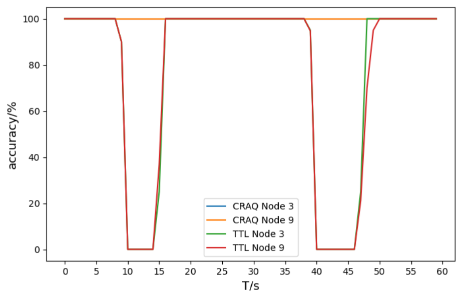

Figure 2: DNS query accuracy at low IP change frequency

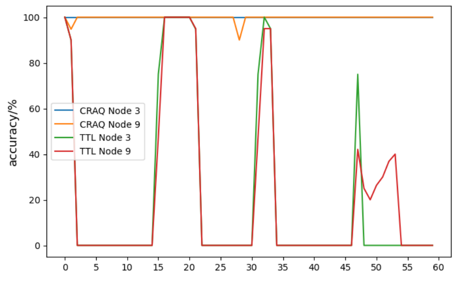

Figure 3: DNS query accuracy at high IP change frequency

However, the accuracy of 9 nodes DoCR at high IP change frequency is not always 100%. We suspect that this is because the IP address updated has not been synchronized to the primary node.

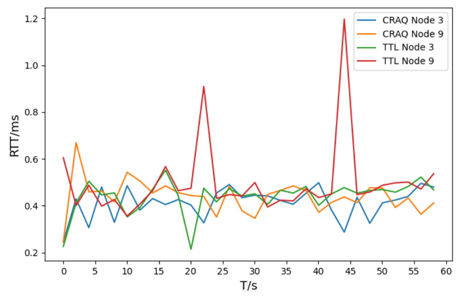

Figure 4: RTT of DNS query

DNS query RTT. Figure 4 shows the RTT curves of DNS queries. We run a 60-second script to send DNS queries to a specific node. The DNS query RTT is similar to that of a traditional DNS server, which means that there is no additional time overhead for read requests to nodes.

IP change request RTT. DoCR introduces additional performance overhead when handling IP change request. As shown in Figure 5, the traditional DNS servers have lower IP change request RTTs and the RTTs are independent of the number of nodes, since the DHCP server only renews the head DNS server on receiving an IP change request. However, DoCR updates each node's DNS items, thus the IP change RTT is proportional to the number of DNS servers. The curves of DoCR in Figure 5 examines the impact of the number of DNS servers. This overhead is acceptable since most workload of DNS transactions would be read.

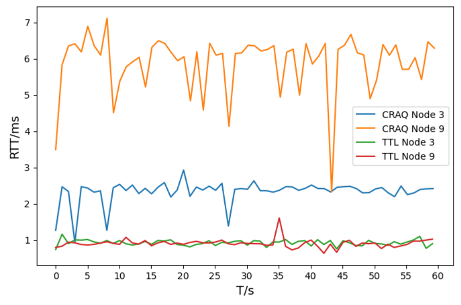

Figure 5: RTT of IP change request

4.3 Fault Recovery

Figure 6 demonstrates DoCR's ability to recover from failure. We show the accuracy change over time for three nodes. Ten seconds into the test, one of the nodes in the chain crashes (i.e. the process is killed). After a few seconds, it recovers (i.e. is activated again) and act as a new node to join the chain. Quickly the accuracy resumes to its original value.

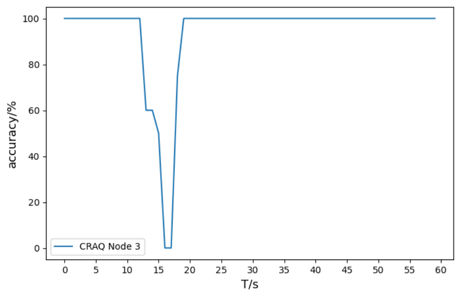

Figure 6: DNS query accuracy change of DoCR during failure

5 CONCLUSION

We implements a new DNS service framework that reaches strong consistency and is compatible with now-existing DNS infrastructure. Our system introduces almost no read overhead and acceptable write overhead. This work could be used in a name zone where a number of servers are mobile.

ACKNOWLEDGMENTS

We gratefully thank our mentor Chenren Xu for the opportunity he provides us to do this work. And Des Preston, who implements an open source version of *CRAQ*.

REFERENCES

[1] Chen Chen, Stephanos Matsumoto, and Adrian Perrig. 2015. ECO-DNS: Expected Consistency Optimization for DNS. In *2015 IEEE 35th International Conference on Distributed Computing Systems*. 256–267. https://doi.org/10.1109/ICDCS.2015.34
[2] Xin Chen, Haining Wang, and Shansi Ren. 2006. DNScup: Strong Cache Consistency Protocol for DNS. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*. 40–40. https://doi.org/10.1109/ICDCS.2006.31
[3] Abhigyan Sharma, Xiaozheng Tie, Hardeep Uppal, Arun Venkataramani, David Westbrook, and Aditya Yadav. 2014. A Global Name Service for a Highly Mobile Internetwork. *SIGCOMM Comput. Commun. Rev.* 44, 4 (aug 2014), 247–258. https://doi.org/10.1145/2740070.2626331

[4] Jeff Terrace and Michael J. Freedman. 2009. Object Storage on CRAQ: High-Throughput Chain Replication for Read-Mostly Workloads. In *2009 USENIX Annual Technical Conference (USENIX ATC 09)*. USENIX Association, San Diego, CA. https://www.usenix.org/conference/usenix-09/object-storage-craq-high-throughput-chain-replication-read-mostly-workloads

[5] Tam Vu, Akash Baid, Yanyong Zhang, Thu D. Nguyen, Junichiro Fukuyama, Richard P. Martin, and Dipankar Raychaudhuri. 2012. DMap: A Shared Hosting Scheme for Dynamic Identifier to Locator Mappings in the Global Internet. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*. 698–707. https://doi.org/10.1109/ICDCS.2012.50