PEER TO PEER CLOUD FILE STORAGE ---- OPTIMIZATION OF CHORD AND DHASH

COEN283 Term Project
Group 1
Name: Ang Cheng Tiong, Qiong Liu

Abstract

CHORD/DHash is a very useful algorithm for uploading data and maintaining data life. In this project we did the performance evaluation for CHORD/DHash, mostly on the structures of files. We also optimized CHORD/DHash with Paxos, a consistency algorithm for updating file on distributed system. By modifying the open source code provided by MIT group, we have successfully enabled the "write" function on CHORD/DHash peer-to-peer system.

Acknowledgement

The source code for CHORD/DHash is downloaded from GitHub at the following URL:
https://github.com/sit/dht/wiki

The authors would also like to thank the technical help of Weide Zhang, an engineer from Microsoft, on development of Paxos.

Table of Contents

List of Tables

List of Figures

2. Introduction

Peer-to-peer systems are distributed systems which are not centrally controlled or have any hierarchical structures. All nodes in the peer-to-peer system have the same capabilities and responsibilities. They allow the sharing of storage resources and also network resources while reducing the operation cost. There are also problems to peer-to-peer systems. Since they are so decentralized, they are hard to maintain and administrate[1~4]. There are also security considerations related with p2p system since all the system level control messages exchanged across the p2p network could be hampered and thus affecting the system behavior. Consistency model for distributed transaction for P2P system is always hard to achieve not only due to the distributed nature of data , but also due to the dynamic behavior of the membership that exist within the P2P system.

Due to the distributed nature of P2P system, the service that need to leverage the overall resources need to be well-designed and efficient. Some systems using Chord[1] and PASTRY[2] can find or insert data in O(log N). Moreover, distributed key/value based file system such as DHash that built on top of Chord tried to address data redundancy by using data fragmentation techniques and uses erasure coding and it also tries to address the fail-over scenario when part of the network is not stable. However, there are also limitations of the system. There is no delete/update operation being supported in current DHash or Chord. Distributed transaction is not addressed in those system since their assumption the data is read-only and address distributed transaction is not trivial since we need to fully evaluate the consistency models that makes sense for P2P system. (provide up to date data freshness in the meanwhile not downgrading the performance of the system)

Our goal in this project is to enhance the existing infrastructure (DHash/Chord) to optimize the stabilization algorithms for routing, to enable “update/delete” operation in DHash and in the meanwhile, tries to evaluate the consistency model that could be applied to “update” operation during distributed transaction. If time permits, we will also tries to implement one or two consistency models in DHash/Chord to support update operation and quantitatively evaluate the performance.

3. Theoretical bases and literature review

There are several tools in the literature for implementation of peer to peer file sharing. One is CHORD. Another implementation is PAST using the PASTRY routing protocol. These 2 technologies both have very good lookup performance timing, O(log N)[1][2].

In PAST[2], each node has a 128-bit node identifier and each file has a 160 bit identifier. These identifiers are hashed to increase security. A minimal routing information is kept at each

node, so that the file is route to the destination in log N hops. These routing information are self-healing, that is, if one intermediate node is down, the node will request information from its neighboring node. These are usually enforced with the usage of keep-alive. However, there is one major disadvantage with the PAST protocol and that is it does not do load-balancing. It storage multiple replica files at different locations regardless of the size. That is, if a very big file requires storage, it might not even get enough space to store it on a node, so the request might be denied.

CHORD uses consistent hashing for data lookup[1]. In consistent hashing, each node (peer) and key (data) is hashed into a CHORD ring including 2^m (m is the number of bits in identifiers) identifiers. Each node n' maintains a routing table with up to m entries, with each entry contains information about the keys in the $2^{(0\sim m)}$ following nodes. With the routing table the time complexity was enhanced to logN. The CHORD ring automatically adjusts the routing tables. When a node joins or departs the ring, CHORD will update the lists of predecessor and successor lists in the routing tables.

Besides CHORD, DHash(distributed hash table) was also proposed by the same group of authors for block storage[5]. In DHash, each file is split into blocks and spread over to several peers. This algorithm balances the load of blocks and decreases the usage of bandwidth because fetching a block requires much less bandwidth than fetching the whole file. To ensure file availability, each block was further split into 14 different fragments, which contains data codes and error correcting codes. The fragments were also distributed to 14 continuous nodes. The block can be reconstructed with any 7 fragments, therefore the availability of the data was significantly improved.

One of the major deficits of CHORD and DHash is that it only supports get and put functions. In other words, users will be able to download and upload file, but are not allowed to edit on the files. CHORD is an "eventual consistence" model which is a very weak consistency model. This model certainly maintains the stability of the ring, but it also disregard any useful modification to the file. To enable editing, another consistency model will be needed.

Another deficit is that although the searching efficiency is improved, there is some extreme cases when the data is in a node that is missed in the first round of searching. Then the searching will be inefficient. In this project, we will optimize the searching efficiency and use a "per-record time order consistency"[6] model to enable the "write" function on a CHORD and DHash based peer to peer cloud file sharing system.

4. Hypothesis

We will focus on two deficits of CHORD and DHash:

1. Improve the efficiency of CHORD ring stabilization after node join/departure using a "two-route" method.
2. CHORD and DHash only allows "put" and "get" files, out goal is to enable the "update" function on existing file. Such operation will involve transaction since multiple data consumer can update the same file simultaneously. Distributed transaction algorithms will be studied and evaluated.

5. Methodology

Generally, for this project we will use the open source code provided by Bakrishnan et. al.[1]. We will use the modified codes to connect 5~10 available machines for result analysis. A visualizer, also provided by Bakrishnan et al.[1], will be used for demonstration how the nodes are connected. Different sample files, including music files, videos files, and text documents of various sizes will be used for testing. The delay in data synchronization (msec), data block availability, data consistency and fault rate will be recorded and compared between original CHORD/DHash and optimized models. The whole project will include the following steps:

1).*Download and modify the open source code for CHORD and DHash to make it runnable*.
The open source code for both CHORD and DHash is available online[1] and we will start with that. However we will need to read the source code for fully understanding and the code needs to be updated to make it work on current versions of operating systems.

2). *To optimize the stabilization efficiency using a "two route" method*.
With CHORD, every time a node join/departure/failure, the CHORD ring will need to update the routing table by requesting each node to check its successors. Then the routing table with m entries will also need to be updated. The total time complexity for these two steps are $O(\log^2 N)$[1, 5]. To improve this, we propose to enhance the stabilization process with a "two route" method. In other words, instead of just checking for its successors, the node will also check its predecessor. For doing this, each node will also need to keep a predecessor list. When any routing table is altered, it then notifies the referred predecessor/successor node to update the table. If It is expected that the efficiency will be improved. However we still need mathematical proof for this part. The algorithm is illustrated at slide 16 of the ppt proposal presentation.

3). *Enhance DHash with "delete/update" operation & Consistency model evaluation for distributed transaction*
As mentioned in the literature reviews, CHORD and DHash only support "put", "get" and "lookup" services. In Bakrishnan's model, they used "eventual consistency" in which there is an unbounded delay in synchronization. This is not ideal if multiple users are writing to the same file. To address this question, we will need to replace eventual consistency with a stronger consistency model, such as per time order consistency[6], in which if a writer writes a lot to a

file, he/she will get the priority to write to the file. Any other users will be treated as readers and can get the updated version from the writer when they make the request. Another possible option is to use the “strong consistency” model[7], in which all updated replicas are consistently copied to all the users. Both of the proposed models needs the support of a central server and will need some modifications.

Implementation

Due to time limitation we finished part 1 and part 3 of the methodology.

1).*Download and modify the open source code for CHORD and DHash to make it runnable*.
The open source code for both CHORD and DHash is successfully downloaded from gitHub. However, because the source code was originally wrote 10 years ago, we have to downgrade the compiler to make it work. The instruction for installation can be found in appendix 1.

3) *Enhance DHash with "delete/update" operation & Consistency model evaluation for distributed transaction*

As mentioned above, we implemented Paxos algorithm on Chord-DHash as a consistency model. In brief, the Paxos algorithm is composed of three kinds of nodes: proposers that can propose values, acceptors that accept a value among those proposed, and learners that learn the chosen value. Each node may take a multiple roles. Paxos is safe for multiple crashes, and can still make progress when less than half of the nodes failed. Thus it is a good candidate for maintaining consistency on CHORD-DHash.

To implement Paxos on CHORD-DHash, the client who wish to make changes is treated as proposers, all live nodes in the CHORD ring with replica of the object file will be treated as acceptors. The clients will be learners. All proposers, acceptors and learners will be initialized as long as it enters the CHORD ring.

The implementation of a Paxos round include the following steps:
1). Propose: Before putting any alteration to the data, the proposers first send out a prepare() request to all the acceptors, including a proposal ID (generated using the timestamp).
2). Promise: When a prepare() request is received by the acceptors, they will send back a promise() message to the proposer(s) if they promise to accept the changes to the data from the proposal. The acceptors are required to promise for the prepare() request with the highest number of proposal ID in a single Paxos round. The proposer will record how many promise() messages it has received; if it has received promise() message from the majority of acceptors, it will start to send an accept() request to all acceptors.
3). Accept: An acceptor will accept the proposed value from the proposer and send an accepted() message to the proposer. The proposer will then record how many accepted() messages it has received. If the majority of acceptors have accepted the value then the value is chosen for the current Paxos round. In each round, only one value can be chosen.
4). Learn: As soon as a value has been chosen, the acceptors will notify every node in the Paxos round. If the current value of the data is the same as the chosen value, then nothing needs to be done; otherwise, the value will be updated to the chosen value.

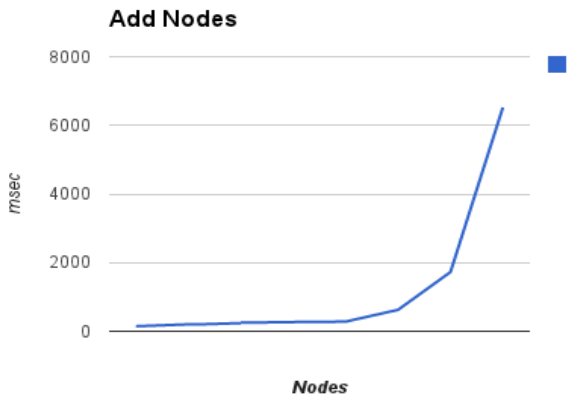
Data Analysis

To demonstrate that Paxos algorithm is successfully implemented on top of CHORD-DHash, we performed several experiments to evaluate the safety and stability of optimized CHORD/DHash with Paxos. All experiments were performed using 6Mbps bandwidth network. The CHORD ring originally contains 0 nodes.

Experiment 1. CHORD/DHash-Paxos basic performance evaluation

We generated the CHORD ring by adding nodes to the CHORD ring using the same IP address. Adding a node from the CHORD ring does not affect the stability of CHORD; The ring re-stabilized in a few seconds (as shown in Figure 1).

Figure1. Add node to CHORD/DHash Ring.



Experiment 2. Chord-Paxos store operation

For this experiment, we uploaded multiple files with the same file name to the CHORD-Paxos ring to see if any conflict would happen. With Paxos, it is required that only one file with certain filename to be stored. In our design, the other clients who did not successfully upload the file for the current Paxos round will be timeout for a random period (0~10 minutes) before proposing another version. The time latency for successfully store file when multiple proposers exist are listed in Table 1. The test file used for this experiment is three JPEG files sized between 60~80 KB. The ring contains 10 nodes.

Table 1. Time Delay for updating files (seconds)

File numbers	First one appear	Last one appear
1	2	2
2	2	182
3	3	455
5	5	417
10	8	633

Discussion

The current project clearly demonstrated that Paxos is useful for maintaining consistency over CHORD-DHash that it only allows for one value to be recorded. However due to the time limit, we still haven't completely implemented Paxos. The biggest issue is that we cannot make changes to an existing file. This is because in CHORD/DHash, each file is divided into 4KB blocks, then each block further divided into fragments, and we were able to locate all the live nodes which has the data of the target file, but was not able to make any change to it. The code was successfully compiled but was having some runtime errors. We will probably need some more time to fix that.

Besides, Paxos itself is having some problems. For example, if two proposers proposing at exactly same time, it may get the same proposal ID because the ID is generated through timestamp. Then the acceptors will consider them to be the same proposal and may promise for both of them in the same Paxos round. In that circumstance, the final chosen value may be from proposer 1 or proposer 2 and the final data may have multiple versions. To avoid the conflict, the proposal ID may use some other random number, such as the node (number + n*rounds); or use timestamp combined with IP address of the proposer. In that way, the ID will be unique.

Another limitation is the instrument and network bandwidth of the data analysis. We used only 10 machines at most, and fixed bandwidth. The influence of different machines, working environment, and bandwidth is not yet evaluated.

Conclusion and Recommendations

The current project showed that Paxos algorithm maintains consistency on CHORD/DHash and can be used for peer-to-peer cloud file storage system. By full integrating Paxos the CHORD/DHash is capable of updating files.

Bibliography

1. Chord: A scalable peer-to-peer lookup service for internet applications. I Stoica, R Morris, D Karger, MF Kaashoek et al. - ACM SIGCOMM., 2001

2. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. A Rowstron,P Druschel - ACM SIGOPS Operating Systems Review, 2001

3. Samsara: Honor among thieves in peer-to-peer storage LP Cox, BD Noble - ACM SIGOPS Operating Systems Review, 2003

4. A survey of peer-to-peer storage techniques for distributed file systems
R Hasan, Z Anwar, W Yurcik et al. - ITCC 2005.

5. A DHT-based backup system. E Sit, J Cates, R Cox - Proceedings of the 1st IRIS Student Workshop, 2003

6. PNUTS: Yahoo!'s hosted data serving platform. BF Cooper, R Ramakrishnan, U Srivastava et al.- Proceedings of the VLDB Endowment. 2008

7. Enhanced paxos commit for transactions on dhts. F Schintke, A Reinefeld, S Haridi et al. - Cluster, Cloud and Grid. 2010

Appendix 1

Installation Guide for CHORD/DHash

The source code was developed 10 years ago. For installation, Both gcc and g++ compiler has to be downgraded to version 4.1 or older. The instruction for installation is as the following steps:

0. install autoconf

   sudo apt-get install autoconf

1. install gmp

wget http://ftp.gnu.org/gnu/gmp/gmp-4.1.4.tar.gz

tar -zxvf gmp-4.1.4.tar.gz

cd gmp-4.1.4

./configure

make

sudo make install

2. install flex and bison

   sudo apt-get install bison flex

3. downgrade your gcc and g++

   sudo apt-get install g++4.1 gcc-4.1

   ls -lrt `which g++`

   ls -lrt `which gcc` // to find out the current symlink path

   usually it should print out /usr/bin/gcc-4.3 and /usr/bin/g++-4.3

   change the symlink:

      sudo rm -rf /usr/bin/g++

      sudo rm -rf /usr/bin/gcc

      sudo ln -s /usr/bin/g++-4.1 /usr/bin/g++

      sudo ln -s /usr/bin/gcc-4.1 /usr/bin/gcc

4. install Berkerley DB

wget http://download.oracle.com/berkeley-db/db-4.5.20.tar.gz

tar -zxvf db-4.5.20.tar.gz

cd db-4.5.20

cd build_unix

../dist/configure --prefix=/usr/local \

--enable-compat185 \

--enable-dbm \

--enable-cxx

make

sudo make install

5. compile sfslite-0.8.16

git clone https://github.com/okws/sfslite.git

cd sfslite

git checkout 0.8.16

mkdir -p ../build/sfslite

./autoreconf -f -i -s

./setup

cd ../build/sfslite

../../sfslite/configure --with-sfsmisc

there is some compile time flag need to change in the makefile of async and arpc lib

vi async/Makefile and change ECFLAGS = to ECFLAGS = -D_GNU_SOURCE vi arpc/Makefile and change ECFLAGS = to ECFLAGS = -D_GNU_SOURCE

and in the build sfslite dir,

make

7. compile chord

git clone git://github.com/sit/dht chord-0.1

git checkout ea40c6690b

mkdir -p ../build/chord

./setup

```
cd ../build/chord
../../chord-0.1/configure --with-sfs=../sfslite --with-db=/usr/local
make
```

Done !