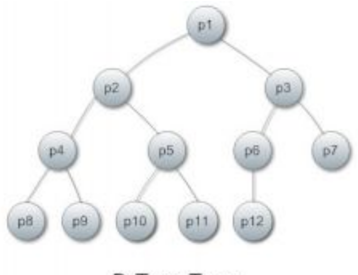
# Performance Analysis

## 1. Background

In order to set up the files into test, we made 120 files of different sizes, which are 10,11,12,...129. Each peer has 10 origin files in their original file folder, as following chart:

| | Peer1 | Peer2 | Peer3 | Peer4 | Peer5 | Peer6 | Peer7 | Peer8 | Peer9 | Peer10 | Peer11 | Peer12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| files | 10-19 | 20-29 | 30-39 | 40-49 | 50-59 | 60-69 | 70-79 | 80-89 | 90-99 | 100-109 | 110-119 | 120-129 |

Since we had 4 topologies implemented in PA2, this time, we choose tree structure to conduct the test, simply because it's more like in the real world., the tree topology is depicted as the following:



B-Tree Type

Also, we use PA2's conf file, the ttl are already assigned there, which enable that every query will reach every node.

## 2. PUSH Analysis

We designed our program by adopting Java Rmi framework, the invalid query result is not necessary to be in the message back, in our case it's the return value of remote invocation. However, in order to calculate invalid query percentage, we made a little change that, adding a state field in query returned message, which is *IpAddr* class in our project. Upon query it, the returned object will carry a state=1, otherwise state=0, in the main function, we abstract those result whose state=1 to download, and whose state=0 to count as invalid message.

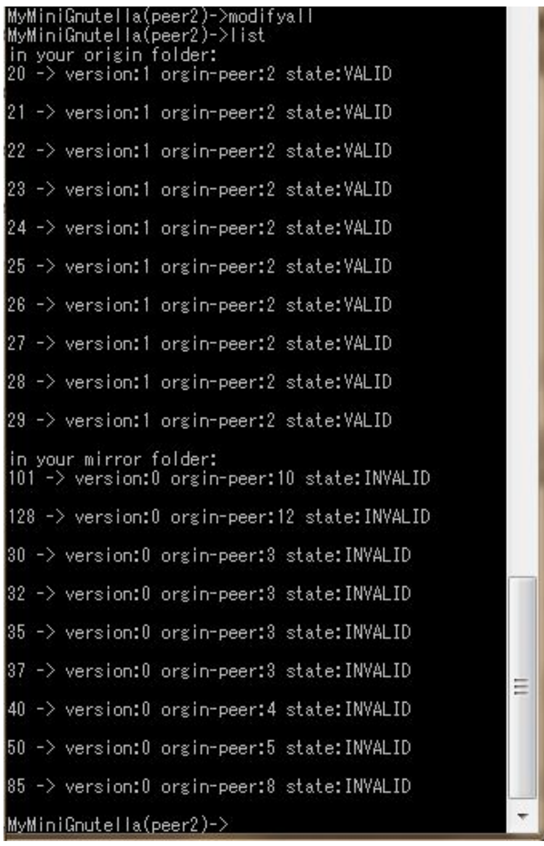
In order to create a situation that invalid and valid copies both exists, we made some new commands to the pull side, which are:

| | |
|---|---|
| *downloadall* | -- randomly download 10 files, which would be put in mirrorfiles folder |
| *modifyall* | -- modify all files in originfiles folder, which will trigger invalidation |
| *testdownload* | -- randomly query 200 times, gather invalid percentage, and print it |

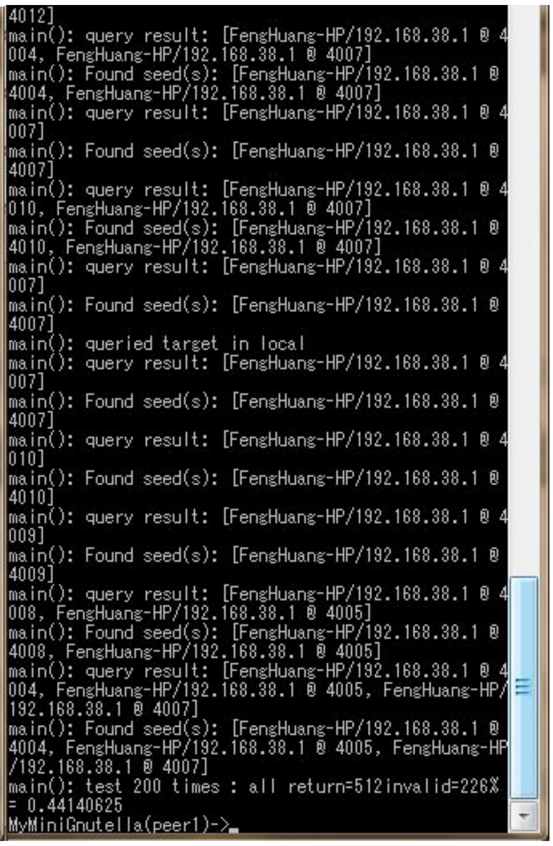
So, the testing procedure is like this:

1. Open all peers, in each peer, run "*downloadall*", except those 1,2,3 and 4 peers for query
2. After run "*downloadall*", run "*modifyall*", after this, most mirror files are INVALID
3. In the particular 1,2,3, or 4 peers, run "*testdownload*", then the statistic would be printed

After doing above, peer2's window shows:



```
MyMiniGnutella(peer2)->modifyall
MyMiniGnutella(peer2)->list
in your origin folder:
20 -> version:1 orgin-peer:2 state:VALID

21 -> version:1 orgin-peer:2 state:VALID

22 -> version:1 orgin-peer:2 state:VALID

23 -> version:1 orgin-peer:2 state:VALID

24 -> version:1 orgin-peer:2 state:VALID

25 -> version:1 orgin-peer:2 state:VALID

26 -> version:1 orgin-peer:2 state:VALID

27 -> version:1 orgin-peer:2 state:VALID

28 -> version:1 orgin-peer:2 state:VALID

29 -> version:1 orgin-peer:2 state:VALID

in your mirror folder:
101 -> version:0 orgin-peer:10 state:INVALID

128 -> version:0 orgin-peer:12 state:INVALID

30 -> version:0 orgin-peer:3 state:INVALID

32 -> version:0 orgin-peer:3 state:INVALID

35 -> version:0 orgin-peer:3 state:INVALID

37 -> version:0 orgin-peer:3 state:INVALID

40 -> version:0 orgin-peer:4 state:INVALID

50 -> version:0 orgin-peer:5 state:INVALID

85 -> version:0 orgin-peer:8 state:INVALID

MyMiniGnutella(peer2)->
```



```
4012]
main(): query result: [FengHuang-HP/192.168.38.1 @ 4
004, FengHuang-HP/192.168.38.1 @ 4007]
main(): Found seed(s): [FengHuang-HP/192.168.38.1 @
4004, FengHuang-HP/192.168.38.1 @ 4007]
main(): query result: [FengHuang-HP/192.168.38.1 @ 4
007]
main(): Found seed(s): [FengHuang-HP/192.168.38.1 @
4007]
main(): query result: [FengHuang-HP/192.168.38.1 @ 4
010, FengHuang-HP/192.168.38.1 @ 4007]
main(): Found seed(s): [FengHuang-HP/192.168.38.1 @
4010, FengHuang-HP/192.168.38.1 @ 4007]
main(): query result: [FengHuang-HP/192.168.38.1 @ 4
007]
main(): Found seed(s): [FengHuang-HP/192.168.38.1 @
4007]
main(): queried target in local
main(): query result: [FengHuang-HP/192.168.38.1 @ 4
007]
main(): Found seed(s): [FengHuang-HP/192.168.38.1 @
4007]
main(): query result: [FengHuang-HP/192.168.38.1 @ 4
010]
main(): Found seed(s): [FengHuang-HP/192.168.38.1 @
4010]
main(): query result: [FengHuang-HP/192.168.38.1 @ 4
009]
main(): Found seed(s): [FengHuang-HP/192.168.38.1 @
4009]
main(): query result: [FengHuang-HP/192.168.38.1 @ 4
008, FengHuang-HP/192.168.38.1 @ 4005]
main(): Found seed(s): [FengHuang-HP/192.168.38.1 @
4008, FengHuang-HP/192.168.38.1 @ 4005]
main(): query result: [FengHuang-HP/192.168.38.1 @ 4
004, FengHuang-HP/192.168.38.1 @ 4005, FengHuang-HP/
192.168.38.1 @ 4007]
main(): Found seed(s): [FengHuang-HP/192.168.38.1 @
4004, FengHuang-HP/192.168.38.1 @ 4005, FengHuang-HP
/192.168.38.1 @ 4007]
main(): test 200 times : all return=512invalid=226%
= 0.44140625
MyMiniGnutella(peer1)->_
```

From the left snapshot, we can see that all peer2's origin files are in version 1, and all its mirror files are outdated. And the right snapshot is after we did "testdownload" in peer 1, our result is 44%.

| | 1 peer | 2 peers | 3 peers | 4 peers |
|---|---|---|---|---|
| Invalid percentage | 0.44 | 0.38 | 0.34 | 0.23 |

From the table, we can see the trend is that the more peers doing modify, the more invalid copies are there, and the more invalid percentage is.

**3. PULL Analysis**

As stated in section 2, we added meta data to see what a query is returning. With the state in a *IpAddr*, we can calculate the percentage of invalid query message.

In order to create a situation that invalid and valid copies both exists, we made some new commands to the pull side, which are:

*downloadall* -- randomly download 10 files, which would be put in mirrorfiles folder

*modifysim* -- waiting exponential distributed time and modify a random file in originfiles folder, repeated this for specific times (200 in the test case)

*testdownload* -- randomly query 20 times, gather invalid percentage, and print it

So, the testing procedure is like this:

1. Open all peers, in each peer, run "*downloadall*", except those 1,2,3 who do query
2. After run "*downloadall*", run "*modifysim*", random modify would be expected in exponential distributed time interval
3. In the peer 1,2,3, run "*testdownload*", then the statistic would be printed
4. Modify ttr in conf file, and lambda in exponential function and re-conduct the test

Output for the test is rather unclear to see, because polling threads never stop, screen is always printing messages, without a stable output. We will just gather statistics in the following chart:

| | ttr = 10s, lambda = 0.1 | ttr=20s, lambda=0.05 | ttr=30s,lambda=0.033 |
|---|---|---|---|
| Invalid % | 0.14 | 0.18 | 0.08 |

Within all three set of ttl and lambda, we tried to match ttl with 1/lambda, that means client is scheduling itself to be as fast as server's modify rate. On the other hand, if we want more reliable of the valid file, ttr can be put less than 1/lambda, we conducted an extra test, which ttr is 10s but lambda=0.05. In this case, server will modify a file in around 20 seconds, but client will poll it every 10 second, the invalid percentage is expected to be lower. It was 0.0244. So we could reach a conclusion that, the ttr : lambda ratio is Inversely Proportional to invalid percentage.

## 4. Comparison

According to push test, a major advantage is that there are very few message flowing between peers, only when a file is modified, the invalidation message may occur a lot, but that is transient. Also, what a peer tells you is always true, which means there is not potential fault because an invalidation takes into affect immediately.
As to Pull, the communication seems to be relatively frequent, this is corelated with ttr value. However, there is a contradicting points because low communication overhead requires high ttr, which increase the rate of getting a false result. The Pull works only better when modifying is sparse and mirror files is few(polling consumes performance)

| | **PUSH** | **PULL** |
|---|---|---|
| advantage | Alway up-to-date | less communication |
| disadvantage | Too much communication if frequently modify | Polling many files consume performance |
| applicability | When modifies are few<br>When consistency is important<br>When peers are few<br>When files are heavily shared | When ttr is large and consistency is not important<br>When files are few |