[change] Distributed File Hosting Systems

Paul Elliott
University of Oregon
paule@cs.uoregon.edu

David Stevens
University of Oregon
dstevens@cs.uoregon.edu

Abstract

FILL IN

1 Project Idea

1.1 Scope/Topic

We're looking into file hosting systems such as Dropbox, which are popular tools for remote file storage and backup, collaborative work, and file sharing.

1.2 Problem

These services/systems work well with centralized storage, but keeping all data at a central server or group of servers:

1. Increases risk of privacy issues
2. Central point of failure

   [FILL IN more from SOUP and elsewhere]

1.3 Solution

One solution to this problem would be to distribute data storage across all users by storing replicas on individual user machines. Such a distributed file hosting system must handle many things: replication strategy, node departures and failures, reads/writes, etc.

To distribute data storage in such a manner requires a distributed and scalable replication strategy.

1.4 Questions

(Given that nodes do not know global state—-and are dealing with a dynamic distributed system–nodes may come and go, nodes may crash, multiple users may work on the same data)

1. Timing/Synchronicity:

   -How do we handle distributed updates of replicas, or rather how do we synchronize updates?

   -How do we synchronize updates

2. Availability vs Network Overhead:

   -How do we handle node failures? How do we update data when a node returns?

   -How do we ensure that a node gets updated/current data?

3. Group membership:

   -How do we keep track of which nodes have replicas?

1.5 Significance

To the best of our knowledge, these strategies have not been comparatively evaluated with respect to distributed file hosting systems.

1.6 Assumptions

Our assumptions for this project include:

1. *Reliable network*: The channels will not fail, or at least channel failures are handled by some error detection scheme (TCP).
2. *Conflict resolution*: Take most recent update
3. *Security*: SHA, PKI [fill in from SOUP]
4. *Event ordering*: vector clock

2 Related Work

(synchronicity/replication strategies–conservative or optimistic–in DS, issues with DropBox, open source revision—file hosting)

2.1 Issues with Centralized File Hosting Systems

(cite dark and exposing papers)

2.2 Replica Maintanence

There has been a considerable amount of work in replica maintenance [1, 3, 2].

Ford: availability in a cloud storage system given different system parameters. NOT decentralized storage environment.

Chun: replica algorithms should 1) focus on durability, a less expensive and more useful goal than availability; 2) durability algorithms must create new copies of data faster than failures destroy them; and 3) more replicas helps tolerate bursts of failures but not increasing likelihood of disk failure. Chun focuses on durability, but we need high availability due to need to keep data in local repository current.

Pu: Discusses various replica control mechanisms which use epsilon-copy serializability (ESR), a correctness criterion that allows asynchronous maintenance of mutual consistency of replicated data. ESR allows inconsistent data to be read, but requires data to eventually converge to 1SR state. "Standard correctness criterion for coherency control such as 1-copy serializability (1SR) are hard to attain with asynchronous coherency control." Pu describes and analyzes these various replica control methods, but does not evaluate them in a specific use case such as [file hosting — revision] systems.

2.3 Synchronization Strategies

(cite textbook, and data replication chapter)

- **Read Once, Write All (ROWA):** ROWA systems allow read operations to merely fetch the requested data from the most convenient location, while write operations are required to update all replicas. Read operations are therefore guaranteed to be correct, but if replica-storing node fails during a write operation, then that write operation will fail. (PE: necessarily true?)
- **Read Once, Write All-Available (ROWA-A):** ROWA-A read operations are the same as in ROWA, but write operations only affect available replicas. This allows writes to subsets of nodes and better handles changes in node availability, but could compromise the correctness of subsequent read requests.
- **Quorum Based (QB):** QB systems allow write operations to subsets of nodes without compromising correctness and consistency. An acting node must collect votes from other nodes until the sum of those votes exceeds predefined read or write quorums. Read and write quorums are defined such that there is always a non-null intersection between any two quorum sets.

3 Design

3.1 Client

Clients are thin–they have local data storage and the broker program, which handles their read and write requests. We implement clients as individual processes with local storage and a database to track timestamps and revision numbers. Clients watch their repository directory for any changes (additions, deletions, modifications).

Upon initialization:

1. update DB
2. start observer
3. connect to broker
4. push entire directory as request

Upon change event:

1. append change to queue
2. send request to broker

Upon broker request ack:

1. dequeue change
2. send change to broker

Upon broker pushback:

1. update DB
2. receive change (lock?)

3.2 Broker

The purpose of the broker in our scheme is to handle read and write requests of the individual clients. Group management: the broker keeps record of all clients participating in the system. A broker keeps its own copy of the revision numbers.

Upon receiving a connection:

1. recv request
2. lock
3. compare request to local DB (revision numbers)

4. if valid respond with ack, else:
    (a) respond with push
    (b) RETRIEVE
    (c) PUSHBACK
5. unlock

Upon receiving client changes:

1. update DB
2. PUSH

3.3 Messages

CONNECT
REQUEST
ACK
RETRIEVE
PUSHBACK
PUSH
(fill in)

3.4 Network

Clients and brokers use TCP stream sockets to communicate with one another.

3.5 Issues Resolved

What if a client dies? Just consider it gone by the system.

What if a client dies after it makes a request? Drop the request.

Client re-emerges (death + reentry) with updated files? Use local database to check timestamps, update local database, and push changes.

How to handle group managament? At the broker.

3.6 NOTES

Watchdog and Asynchchat = observer pattern/asynch commun.

4 Evaluation

[THIS ALL NEEDS TO BE UPDATED]

1. Testing Platform:

2. Parameters: Centralized broker vs decentralized broker.

   Different levels of node churn.

   Different replication strategies.

3. Metrics:
   - Available:
     -network overhead?
     -failed pull attempts?
   - Reliable:
     -number of successful up-to-date pulls?

4. Scalability:

5 Timeline

1. This weekend: complete literature review (synchronicity/replication strategies in DS, issues with DropBox, open source revision—file hosting)

   -Each of us: research 1 of each

   -Fill in 1 paragraph summary of any literature, add any ideas to notes

2. Week 8: Design
   (a) Build client
   (b) Implement storage
   (c) Create network definition (graph, edges are cost)
   (d) Implement central broker (define policies)
   (e) Implement Distributed broker

3. Week 9 (Thanksgiving): Testing/Evaluation

   TO CODE:
   (a) Client: local storage (sqlite)
   (b) Client: handle init (update db, start observer, push all)
   (c) Broker: local storage (mysql? sqlite?)
   (d) Broker: handle request (either accept/respond or pushback current)
   (e) Broker: request queue
   (f) TIMERS/TIMEOUT
   (g) Centralized server?
   (h) Testing environment

4. Week 10: Presentation, Write paper

5. Final Exam Week: Write paper, turn in paper, have multiple celebratory beers

References

[1] B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. Kaashoek, J. Kubiatowicz, and R. Morris. Efficient replica maintenance for distributed storage systems. In *Proc. of NSDI*, volume 6, pages 225–264, 2006.

[2] D. Ford, F. Labelle, F. Popovici, M. Stokely, V. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation*, 2010.

[3] C. Pu and A. Leff. *Replica control in distributed systems: as asynchronous approach*, volume 20. ACM, 1991.