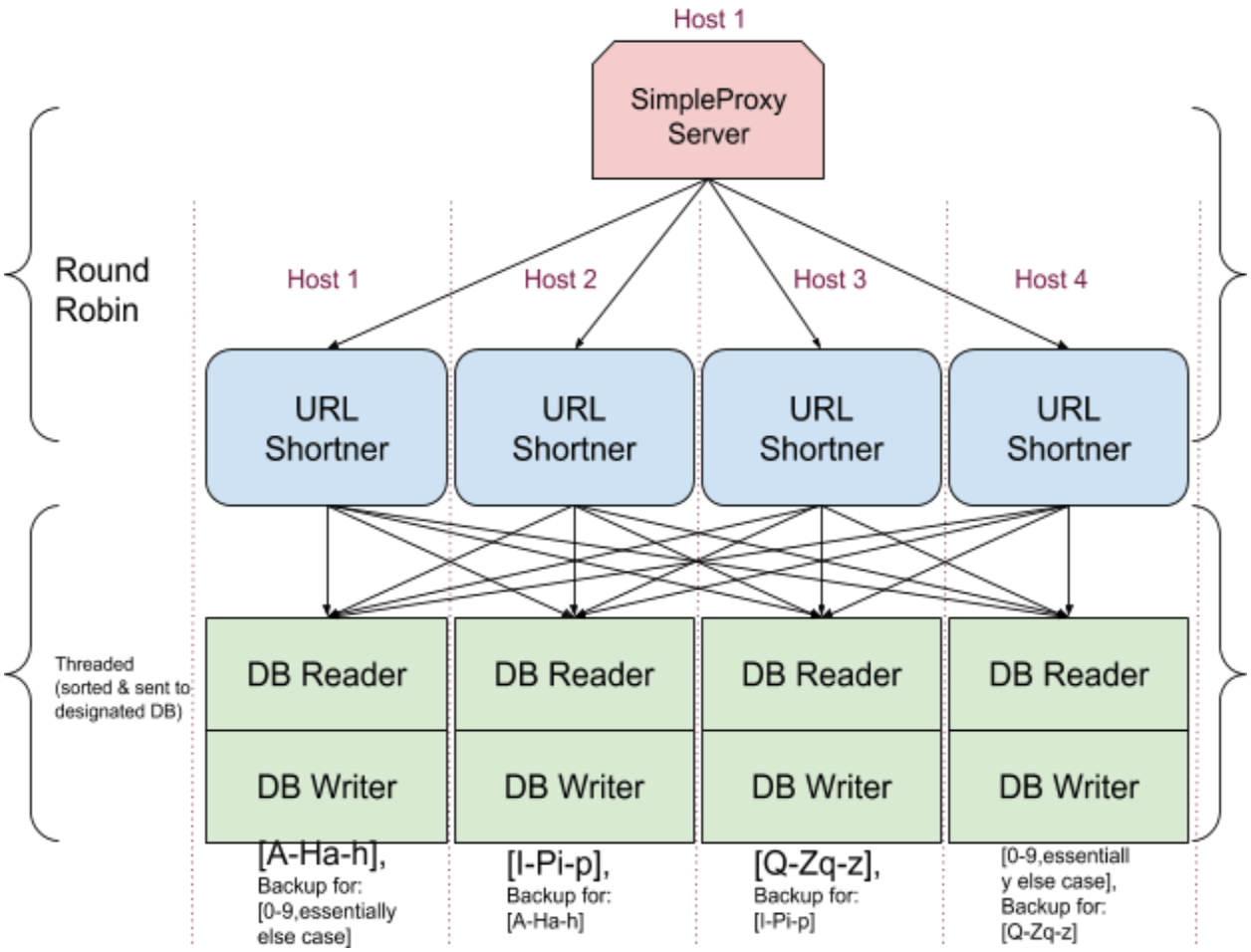
Architecture



One machine runs the proxy server (SimpleProxyServer PORT 2020), which is assumed to be functional at all times in order for the system as a whole to be available for users.

The proxy redirects to our (URLShortner PORT 4040) servers, running on all four of our machines. It has a thread pool of designated size 4 (but is changeable within the code for scalability). Proxy to URLShortners is round robin; we are not taking into consideration the workload of a URLShortner in the decision making process.

The URLShortner connects to one of the DB Controllers, which has a thread of the writer (PORT 5050) and reader (PORT 3030), corresponding to one of the following sections: [A-Ha-h] [I-Pi-p] [Q-Zq-z] [0-9,specialChar,ELSE]. Each machine (let's call it N) has a database.txt with "key value pairs" for it's designated section, as well as a databaseBackup.txt, which is a copy of machine N-1's database.txt.

Caching is also implemented among DB servers which saves the most recent reads/writes. (Never inconsistent since each character goes to the same DB) In the case that

DB X is now handling TWO partitions (since X-1 went down) we clear the cache each time we swap between DB and DBbackup

Running The System

Starting up the URLShortner is through the heartbeat.sh script, where X is the machine number in the lab, 09 <= X <=27:

$ heartbeat.sh X

It calls the startURLShortner, startSimpleProxy, and startDatabaseController for the user on the designated hosts X, X+1, X+2, X+3, and creates the necessary database files on /virtual/$USER for each host and sets permissions.

Our heartbeat.sh script checks for the designated ports of each service every 15 seconds. If not available, the service on the machine will be restarted.

As well, the script copies each of the database text files on host X onto the host X+1 as databaseBackup.txt (every 15 seconds as well).

Requests can be made from the command line with curl or one of the test files, as well as on the browser with the IP of the host running SimpleProxyServer (hostX), or using a firefox opened from the host X's terminal with http://localhost:2020/ for writing, or http://localhost:2020/[SHORT] to read, where [SHORT] is the shortURL.

When the service is to be terminated, the user can call the stop file, to kill all of the processes on every host machine(X, X+1, X+2, X+3).

$ sh ./stop X

Testing The System

./performanceTesting/loadTest

- Runs the ab test

./performanceTesting/readTest.py

- GETs for same short url - good for demonstrating how effective our cache can be when trying to get the same url over and over

./performanceTesting/writeTest.py

- PUTs for random short and long urls - stresses URLShortner to max by sending out write requests that are different every single time; testing if we receive back server errors

./performanceTesting/readTestRandom.py

- Same as readTest GETs format but the shorturl is like writeTest where each request is random

./performanceTesting/readwriteTestAVG.py

- 4 x readTest - random
- 4 x writeTest - random

- 4 x writeTest - hardcoded puts
- 4 x readTest - hardcoded reads
- Filling the cache (size 16) with attempt measure random amounts of reading and writing random shorts but in between reading short urls known to exist in the cache. Cycles 625 times to total 10k requests

Writing on the browser at localhost:2020 also functional, even if services may go down.
Testing included the killing of services on machine(s) during tests above with no server errors.

Analysis

CAP Theorem:

Consistency [eventual] - there is no full consistency, only eventual consistency. In that sense, our system backs up the database text files onto another machine every 15 seconds. Therefore, we can read from the backup text file if the machine hosting that section of the database goes down; however, we have no guarantee for it being the latest data. If we write to a section of the database and the service on said host X is down, then we will write to the backup file on host X+1.

On restart to the database host X, we will copy back the “backup” text file on host X+1 onto host X, and resume function. This can be inconsistent in terms of data loss; if the host goes down and we have a difference in the main and backup text files, we lose all of the changes on the main text file. This is a battle of having a short time between copying the database text file for more consistency in the data, versus gaining speed from not copying over as often but potentially losing much more data.

Availability [yes] - We have availability because the user will get a response even if the designated host for a database section is down. We have a copy of the database on a secondary host (X+1) such that if host X is unavailable, we can still get a (potentially old) data copy of the long url retrieval on a read. Similar to read, writing is also available to the user, as they would be writing to the backup host. There is only an issue if BOTH hosts X and X+1 containing the main and backup text files of a database section go down since we only keep 2 copies of each partition.

Partition Tolerance [yes] - We have multiple partitions, and we have replication of the databases (a main copy and a copy of the database on host X+1). So, from this partitioning, even if one of the databases is unavailable, we still have a (potentially old) copy of the database, and to the user it looks like the system is still functional. Therefore, even if something is downed, we still have a functioning system that takes in responses and handles them appropriately (returns a url or not found page).

Scalability:

- URLShortner uses a thread pool from ExecutorServices, which is default set to 4 because of the lab machines; however, the pool size can be increased within the code (there is just not any payoff in the case of the machines for this assignment)
- More URLShortners and DB readers/writers if we have more hosts; able to create more threads to access the DB
- Partitioning the DB more on more hosts - we sort requests by the first character in the short URL, and each database section on a host only takes in a set of starting characters, so if we have mass requests for chars close to each other, there is less likelihood of all of the requests going to one machine
- Bottlenecks would include time spent to decide where to send requests

Latency:

- With the cache we are retrieving the last cached items before trying to read from disk, so the runtime is significantly faster. This way ./loadTest or readTest.py (which reads from one URL) can run 1000 requests with latency of 30ms
- Our cache can be expanded to hold more recent requests, but then we would be using more resources
- When deciding what Database to send the write/read request to, the server has to find the starting character, and then send to the designated read/write database server the given input to read/write - this could add a small delay each time
- Results for our tests are in the “Throughput” section

Availability:

- Service Failure:
    - If a service on a singular host goes down, we are safe. Unless we have some sort of combination that kills the service on both the main host X and host X+1 (holding the backup of host X), then that can lead to unavailability for that partition of the database
    - If the proxy goes down, the entire url shortner service goes down - we are assuming that the proxy is stable and to be run on only one host
    - As long as one of the URLShortner services on any of the hosts is available, we are functional with possible inconsistent data
    - If one of the database services goes down, we still have a backup of the database text file on host X+1
    - Heartbeat.sh script checks for downed services and restarts them on that designated host every 15 seconds
    - If a service cannot be restarted on a host for any reason, the backup database text file will continue to be read from/written to and act in place of host X (until the service on host X can be revived)
- Host Failure:
    - As noted above under Service Failures, in (almost) all cases, we are safe to lose a Database or URLShortner service on a machine
- Storage Failure:

- Database files split up across all hosts by the starting character of the short URL to reduce storage issues on all machines.
- In case of Storage failure we also have backups of X on X+1

The complete system remains available and consistent so long as:

i. The load balancer (SimpleProxyServer) is available and functioning - this is a must and without it the system as a whole is down for users
ii. The URLShortner is available and functioning on at least one of the designated hosts to handle requests
iii. All Database services are available and functional as long as at least one of two machines are running the service for a section of the database:
    - The main host X running the reader and writer for a section of the database
    - The backup copy of the database file (databaseBackup.txt) on the host X+1

Durability:

- Writing is immediate, so there will be limited data loss, since, for example, we aren't writing a buffer
- The database is partitioned into 4 sections: [A-Ha-h] [I-Pi-p] [Q-Zq-z] [0-9,specialChar,else] each on some host X, and is backed up in a seperate text file on machine X+1
- If a host X goes down, then the designated section for host X is available from the text file stored in X+1 to read and write to; once host X has its services restarted, then we copy back the backup text file onto X to preserve changes/later writes made during its downtime
- We have a timer of 15 seconds so that backup is often, and if machine X goes down, then we have more recent data stored on X+1; however, it means that any changes on X that have not been scp'd to X+1 before going down is lost

Consistency Guarantees:

- If one of the database services goes down, then we are not guaranteeing consistency in our returns to the users for the long urls -> We have an older version but it may be inconsistent within 15 seconds
- As noted under "Durability", we have some forms of backups to the database text files; however, there may have been loss if a change was made to the main file that was not copied over yet. This data is completely lost, and we also lose our cache for that host
- If we are writing while some host X is down, we have read and write ability to the backup on X+1, which is then pushed back to host X for some form of *eventual consistency*
- Any data older than 15 seconds will return the most recent data

Fault Tolerance:

- Losing a URLShortner will just eliminate it from the proxy's round robin until it is available again - service still available
- If a single service on any host fails, the heartbeat.sh script will restart the service; heartbeat checks for downed hosts every 15 seconds
- A downed database service on a machine just throws the request to the machine hosting the backup database text file

Throughput:

Best loadTest result:

This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests

Server Software: Java
Server Hostname: localhost
Server Port: 2020

Document Path: /0000000000000000000000000000000000000000
Document Length: 53 bytes

Concurrency Level: 10
Time taken for tests: 2.095 seconds
Complete requests: 1000
Failed requests: 0
Non-2xx responses: 1000
Total transferred: 200000 bytes
HTML transferred: 53000 bytes

Requests per second: 477.33 [#/sec] (mean)
Time per request: 20.950 [ms] (mean)
Time per request: 2.095 [ms] (mean, across all concurrent requests)
Transfer rate: 93.23 [Kbytes/sec] received

Connection Times (ms)
              min mean[+/-sd] median   max
Connect: 0   0   0.0   0   0
Processing: 3   21   1.6   21   27
Waiting: 3   21   1.6   21   27
Total: 3   21   1.6   21   27

Percentage of the requests served within a certain time (ms)
  50%   21
  66%   21
  75%   21
  80%   22
  90%   22
  95%   23
  98%   25
  99%   26
 100%   27 (longest request)

readTest:

# of Requests: 10000
Total Time: 85.98244285583496 s
Time per Request: 0.008598244285583496 s

writeTest:

# of Requests: 10000
Total Time: 87.46469521522522 s
Time per Request: 0.008746469521522521 s

readTestRandom:

# of Requests: 10000
Total Time: 86.48282814025879 s
Time per Request: 0.008648282814025878 s

readwriteTestAVG:

# of Requests: 10000
Total Time: 86.87825918197632 s
Time per Request: 0.008687825918197632 s