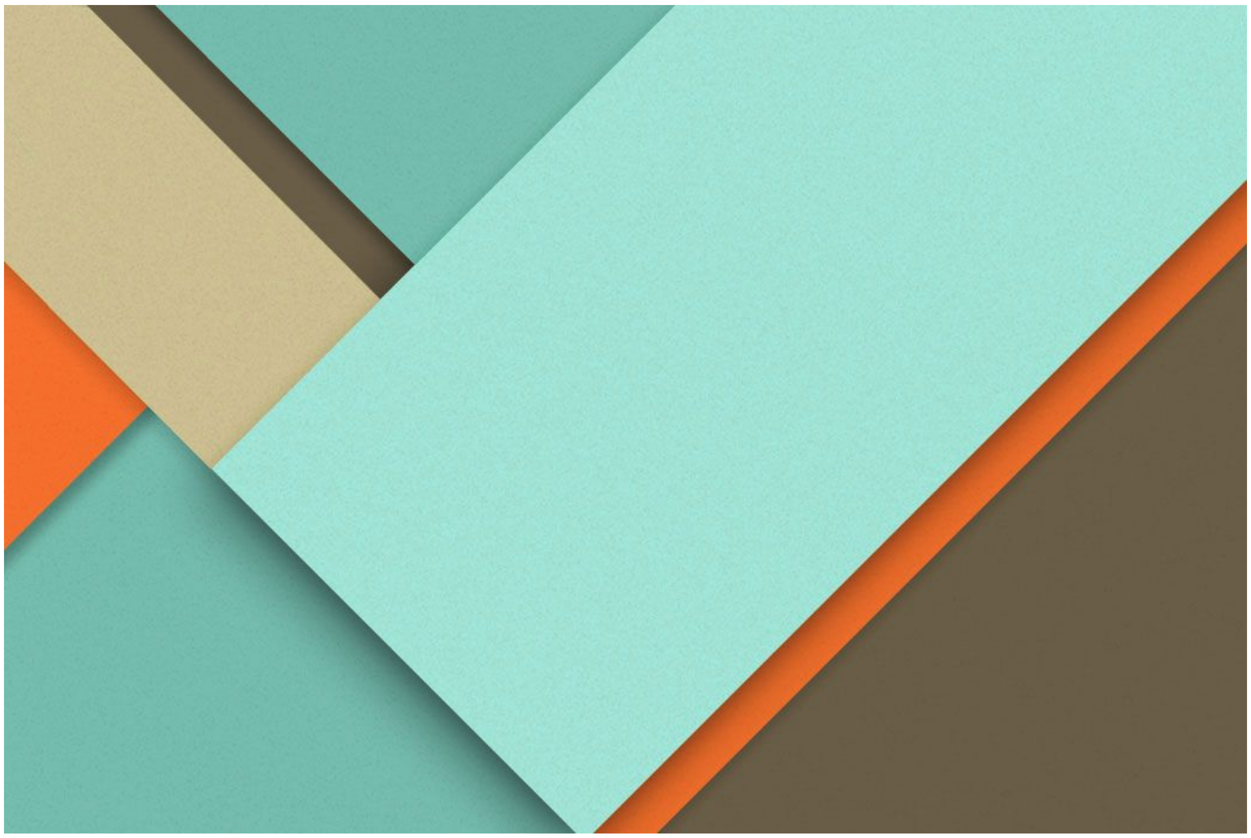

Distributed Directory Service

17.03.2019

Group 7
Vishesh Agarwal
Vishal Gupta
Nishant Somy
Lakshya
TA: Gulab Arora

Overview

The project aims to realize a distributed directory service. We aim to deploy a subset of the functionality described in the OSI LDAP standard, but hope to get down the exact implementation working according to our own terms.

User Features

We plan to implement the following user features for the three user categories:

1. **Directory Admins:**
   I. Be able to specify and modify directory schema.
   II. Manage access controls over the directory data.
2. **Publishers:**
   I. Modify existing data.
   II. Add entries.
   III. Delete entries.
3. **Subscribers:**
   I. Read data of an object, if they know its name.
   II. Search for objects, filtered by certain rules, within a specified scope.

System Features

We expect the following system features out of our service:

1. **Fault Model:**
   I. One crash fault tolerance
   II. One link failure tolerance:
   Hopefully, we will be able to achieve at least a somewhat degraded service. We just don't want to let the link failure go unhandled.
2. **Consistency Model:**
   → Eventual Consistency with a time constraint:
   The published updates must become consistent across the database within *some* fixed number of read requests.

Design Questions

1. Where to store complete directory:

a. All eggs in one basket (single node):

Pro:

I. Easy to maintain.

II. Don’t need to go any further to search for data.

Con:

I. Highly resource intensive. Impractical for large directories.

II. Longer search time as no true parallelization (can only achieve parallel search via threading).

b. Distribute among multiple nodes:

Pro:

I. Distributed Load. Truly parallel search for data possible.

II. Faster search compared to single node if there is lots of data.

III. Scalable.

Con:

I. Need to reroute requests, probably need to remember what data might be where.

II. For smaller data, may perform slower due to redirection overhead.

III. Supporting structural changes in DIT will become very complicated.

CHOSEN OPTION: Distribute tree data over multiple nodes.

NEED TO ALSO THINK:

How to partition the tree and what servers must be connected to what servers?

Should the network structure be static or dynamic (depending on request frequencies)?

2. **How to reroute requests:**

   a. **Chaining:**

      Pro:

      I. Less work for DUA.

      II. Access is secure amongst directory servers.

      Con:

      I. More work for DSA.

      II. Need to remember forwarded query contexts until solved.

      III. Requesting faraway information repeatedly via the same chain is as inefficient as things can be.

   b. **Referrals:**

      Pro:

      I. Less work for DSA.

      II. Requesting faraway information repeatedly becomes fast as you know where to go once you ask.

      Con:

      I. More work for DUA.

      II. Users get to know more about the directory structure. Might introduce some security issues.

CHOSEN OPTION: Some hybrid of the two would be preferable.

3. **Who detects the fault?**

   a. **Centralized Node:**

      Pro:

      I. It knows about everything. Can handle issues like link failure better.

      II. Can resolve issues that might affect multiple nodes at once.

      Con:

      I. High communication traffic to central node.

      II. Everything needs to go to one guy, so fault resolution will be slower.

      III. Tougher to prove if it can handle everything.

b. **Local neighbours:**

Pro:

I. No overhead of getting all information anywhere. Everything is locally resolved.

II. If we can prove it works for one neighbour pair, the unit can be repeated and is guaranteed to handle all sorts of faults.

III. More scalable.

Con:

I. Link failures are tough to handle as both neighbours believe the other is dead while both are alive.

II. May not be able to handle more widely disruptive faults using only local resolutions.

CHOSEN OPTION: Leaning toward local fault resolution.

Will try to figure out how to handle link failures.

4. **How many replicas?**

a. **Two:**

Pro:

I. Can handle single crash fault.

II. Consistency issues are easier to resolve.

Con:

I. Data access points may be too far from where data is kept. So, longer chains, and more latency.

b. **Many:**

Pro:

I. Data access will be faster if data is distributed evenly around the globe.

Con:

I. Consistency gets tougher to maintain across multiple replicas.

CHOSEN OPTION: Will try to keep few (three-four) replicas, but get the data dynamically closer to where they are being accessed. Can also try keep number of copies dynamic, based on request frequencies.

5. **How to maintain data consistency?**

    a. **Passive Replication:**

    Pro:

    I. Consistency is always maintained.

    II. Simpler to implement.

    Con:

    I. Writes are slower.

    b. **Active Replication:**

    Pro:

    I. Flexibility. Lots of ways to maintain consistency.

    II. Can be made as fast as needed.

    Con:

    I. Implementation is more complex.

CHOSEN OPTION: Passive Replication as we expect very few writes.