Phase 2 - Design decisions

In this document the design and implementation of Agent-Smith, a distributed storage network, based on the pithos architecture, will be discussed.

More specifically, the design and implementation of `phase-2`, the Foundations of Agent Smith's service layer, is discussed.

Phase goal

The goal of phase-2, was to design and implement the fundamental service layer of Agent Smith. Features implemented in this phase include:

- Session control
- Application structure & flow
- Super Peer selection
- Dynamic Grouping

Component/Feature implementations

The following sections will discuss each component/feature in more detail

Features implemented as part of `phase-2`:

- Group Storage
- Grouping mechanism
- Functional Peers
- Functional Super Peers
- Leader selection
- Directory server updates
- Improved system control
- Improved fault tolerance

Application structure & flow

Current Application Structure:

Pithos was designed to fulfill all use case requirements of `responsiveness`, `fairness`, `reliability`, `scalability` and `security`. Agent Smith aims to follow and adhere to these core requirements, by making it part of the storage networks fundamental functionality.

In phase-2 of design/development, 4 of the above mentioned requirements were focused on, namely:

- Reliability
- Scalability
- Fairness
- Responsiveness

the final requirement, *Security* will be introduced in the following development phases.

Reliability

As addressed in `phase-1` of development, reliability is achieved by storing all objects on **overlay-storage (DHT)**.

In a fully connected network, where all nodes are connected to all other nodes, every node is one hop away from every other node. This solution is, however not scalable. In order to achieve a scalable single hop architecture, it was decided to group users in the virtual world. User groups are then fully connected, which allows for highly responsive group interactions.

As part of `phase-2`, **Group-storage** was introduced, which groups peers together and replicates all data across each node in a group. To simplify development, groups are currently limited to a fixed size of 3 peers per group.

For grouping, the *Zookeeper** grouping mechanism is used.

Super Peer Selection has been introduced to act as a master node for each group.

For Super peer selection, the *Zookeeper** leader election mechanism is used.

> Zookeeper, acts as Agent Smith's Directory server, which helps peers connect to the closest Super peer, supplying it with the correct IP/Port combination.

Scalability

Scalability is achieved within the Pithos architecture by,

- Having Group sizes Smaller or equal to total network size
- Peers do not service requests outside of it's group
- For out-of-group requests, the overlay storage is used

As scalability is a crucial for any distributed system, careful consideration has been taken to ensure that components are sufficiently scalable. During `phase-2` the following features aid towards achieving scalability within the system:

- Dynamic Grouping

Fairness

Fairness is achieved within the Pithos architecture by using both group storage and overlay storage.

During `phase-2` the following features aid in achieving fairness within the system:

- Group storage
- Functional roles for peers & super peers

Responsiveness

In Pithos, responsiveness is achieved by grouping users into fully connected groups, using group-based distance-based storage to distribute objects and using replication to enable the parallel retrieval of objects.

During `phase-2` the following features help towards fairness within the system:

- Group storage - group based storage of objects

Module Types - Systems roles

The Pithos architecture define three main modules in the system namely,

- Communicator
- Peers
- Super Peers

The **communicator** module acts as an event hub, that sends all communications to the upper and lower layers on behalf of the other Pithos module.

The **peer** module is responsible for group and overlay storage.

The **super peer** module is responsible for group membership and object repair.

During `phase-2` the following modules were introduced:

- MainController
- Peer
- Super Peer

Main Controller

During this development phase, the concept of a **communicator** was introduced in the for of the **Main controller** class. As it stands, this controller has no other functionality than to assign a role to the node, connecting to the Storage network. The only two roles that are currently defined are **Peer** and **Super Peer**. The logic behind this role assignment is extremely trivial, thanks to Zookeeper's *Grouping* & *Leader Election* mechanisms. The controller queries the Directory Server on startup, if the id of the leader is the current assigned ID, this node will become the group's **Super Peer** and otherwise it will become a **Peer**.

Peer

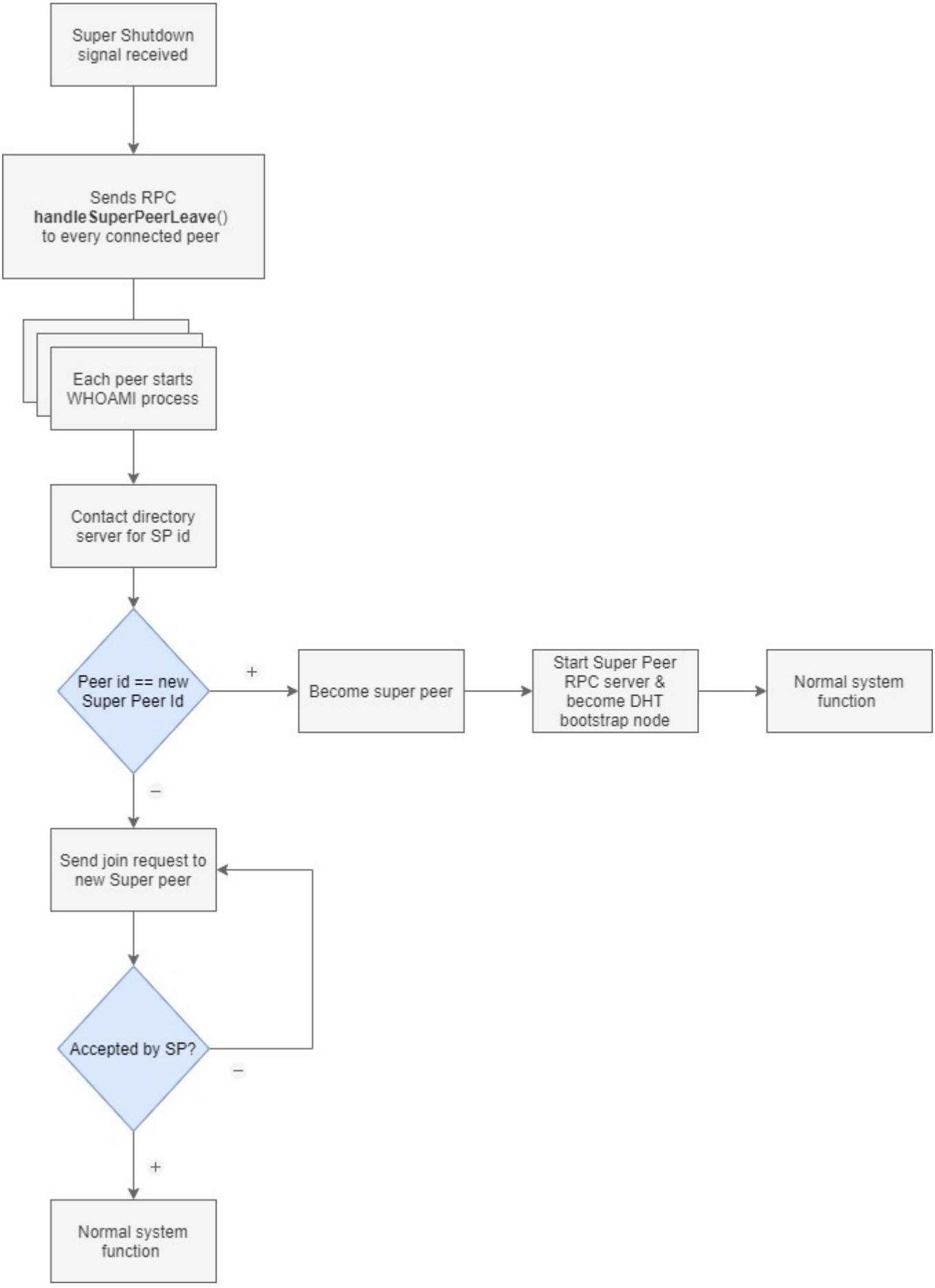
During this development phase, the previously introduced **peer** class has been expanded on. In it's current state the peer class contains the following modules & logic:

- Peer Storage
    - Local storage
    - Overlay storage
    - Group storage
- Directory server client
- Group Ledger
- gRPC Peer server - functionality:
    - Handle Add peer requests
    - Handle Remove peers requests
    - Handle Super peer leave requests
    - Assign a new role if super peer leaves/fails

- Super peer gRPC client - functionality:
    - Join group
    - Add object reference
    - Leave mechanism
    - Notify peers that a node joined the group

One of the challenges of this phase was to determine how a new Super peer will be elected and started, if the previous Super Peer either leaves or fails. To solve this problem, the pithos architecture introduced the concept of *redundant super peers*, which is not yet implemented in Agent Smith.

To solve this in the current state, The following logic was introduced:



Super Peer

During this development phase, the **Super peer** class has been introduced. In it's current state the super peer class contains the following modules & logic:

- Peer Storage
  - Overlay storage
- Directory server client
- gRPC Super Peer server - functionality:
  - Update group ledger
  - Update group name
  - Update list of peers & peer grpc clients
  - Handle join requests
  - Handle leave requests
  - Object repair (wip)
  - Migrate requests
  - Add object references, update ledger
  - Broadcast when new peer has joined the group
  - Handle shutdown gracefully
- peer gRPC client(s) - functionality:
  - Join group
  - Add object reference
  - Leave mechanism
  - Notify peers that a node joined the group

Super Peer Selection

A simple way of doing leader election with ZooKeeper is to use the SEQUENCE|EPHEMERAL flags when creating znodes that represent "proposals" of clients. The idea is to have a znode, say "/election", such that each znode creates a child znode "/election/guid-n_" with both flags SEQUENCE|EPHEMERAL. With the sequence flag, ZooKeeper automatically appends a sequence number that is greater than any one previously appended to a child of "/election". The process that created the znode with the smallest appended sequence number is the leader.

Dynamic Grouping

Another function directly provided by ZooKeeper is group membership. The group is represented by a node. Members of the group create ephemeral nodes under the group node. Nodes of the members that fail abnormally will be removed automatically when ZooKeeper detects the failure.

Technical dept for phase-3

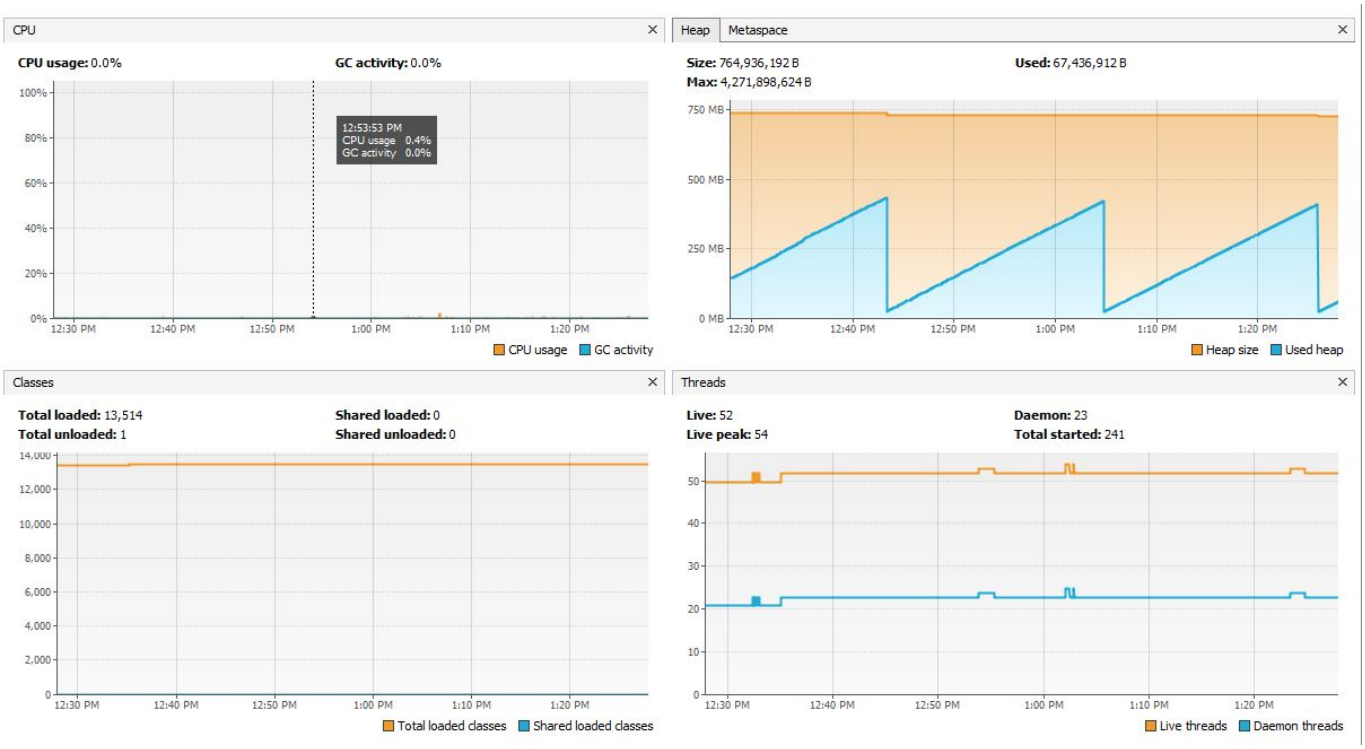
The necessity of the following features were made apparent during `phase-2` of development but not addressed and will be addressed in up-coming phases:

- Redundant Super peers
- Dynamic group assignment
- Dynamic group sizes
- Zookeeper Leader Electors per group
- Introduce an event hub as communicator to replace **Main Controller** class

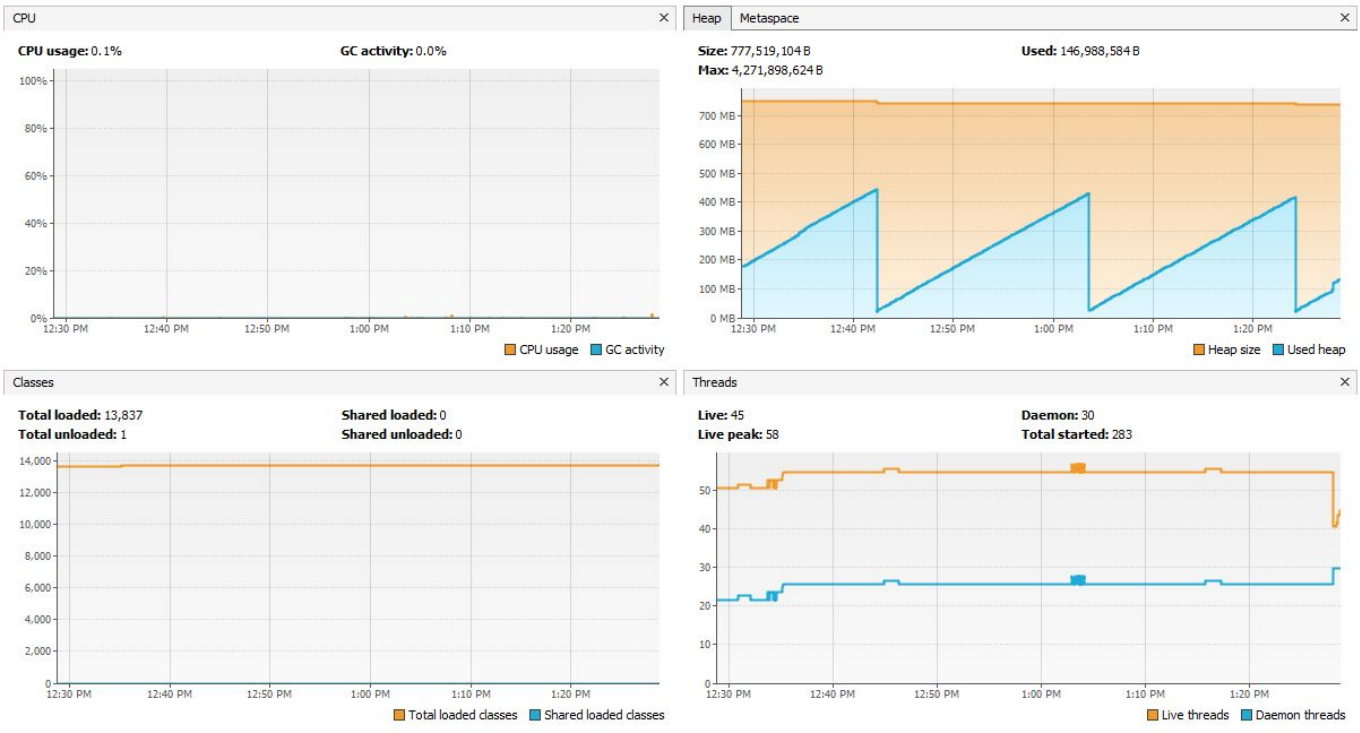
- Implement repair of objects
- Implement migration of peers
- Implement secure gRPC connections (enable TLS/SSL)

Memory Analysis

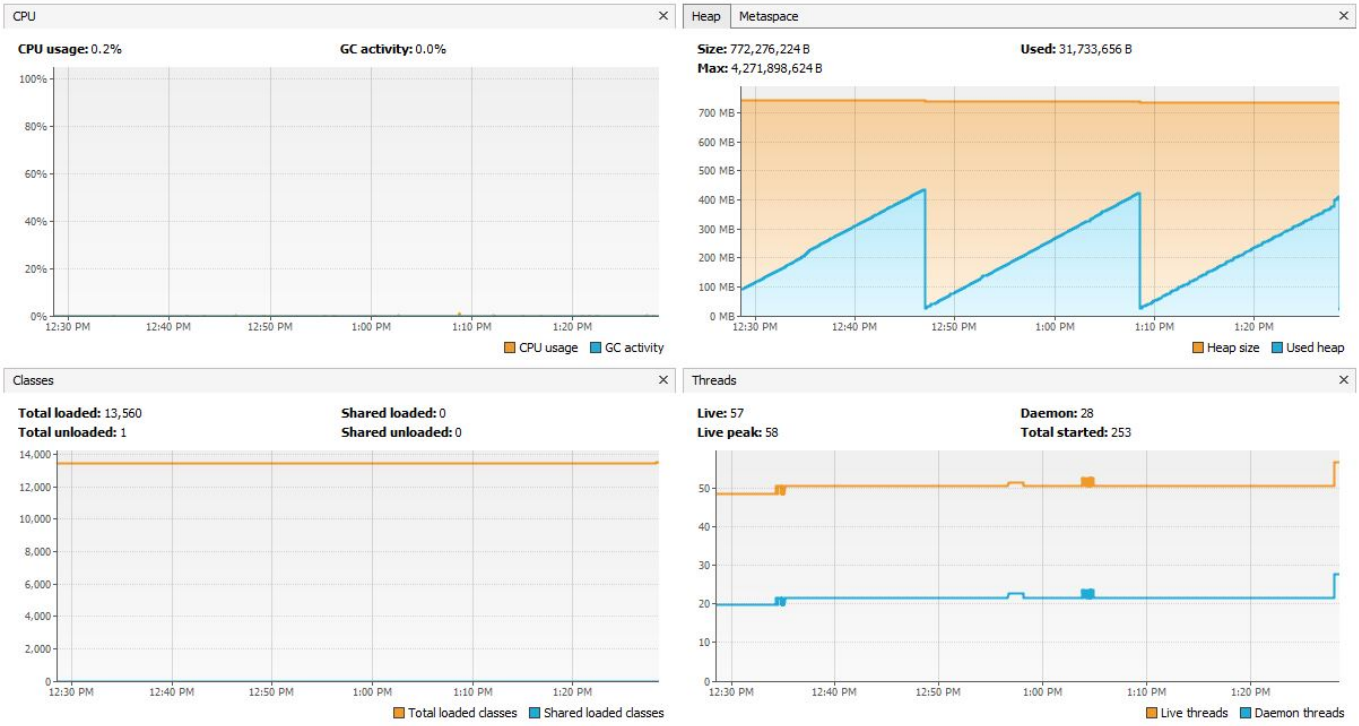
Super-peer



Peer-1



Peer-2


CPU
CPU usage: 0.2%
GC activity: 0.0%
100%
80%
60%
40%
20%
0%
12:30 PM
12:40 PM
12:50 PM
1:00 PM
1:10 PM
1:20 PM
CPU usage
GC activity
Heap
Metaspace
Size: 772,276,224 B
Used: 31,733,656 B
Max: 4,271,898,624 B
700 MB
600 MB
500 MB
400 MB
300 MB
200 MB
100 MB
0 MB
Heap size
Used heap
Classes
Total loaded: 13,560
Shared loaded: 0
Total unloaded: 1
Shared unloaded: 0
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
Total loaded classes
Shared loaded classes
Threads
Live: 57
Daemon: 28
Live peak: 58
Total started: 253
50
40
30
20
10
0
Live threads
Daemon threads