1. Statement of the Problem

In this project, we aim to replicate the core functionality of Napster, a file-sharing system, with a key modification: implementing **on-demand replication** (increase replicas for high demand files and reduce the replicas if the demand drops) to improve fault tolerance and load balancing. Traditional peer-to-peer (P2P) file-sharing systems often suffer from two major problems:

- **System failures**: If a peer hosting a file goes offline, the file may become unavailable.
- **Overload on popular peers**: Some peers may receive an excessive number of file requests, leading to increased latency and network congestion.

Our goal is to address these issues by dynamically replicating files across multiple peers based on demand and system status. By replicating popular files on additional peers and ensuring balanced load distribution, we can improve the overall resilience and scalability of the system.

2. Important Papers/Material

1. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1040912
2. https://research.vu.nl/ws/portalfiles/portal/2450843/scc.04.pdf
3. https://dl.acm.org/doi/pdf/10.1145/1966445.1966455
4. https://www.researchgate.net/publication/30499478_The_Rise_and_Fall_of_Napster_-_An_Evolutionary_Approach
5. https://arxiv.org/pdf/cs/0402018

3. Scope of the Project

The project's main focus is to replicate Napster with a **dynamic, on-demand replication mechanism** that addresses file availability and load balancing. Instead of statically replicating files at fixed intervals or with a fixed replication factor, we propose a system where:

- Files are replicated dynamically based on their access frequency. Highly requested files will be replicated across multiple peers to ensure they remain accessible, even if some peers go offline.
- Load balancing is achieved by redistributing file requests across peers storing replicated copies, preventing overload on any single peer.

Use Cases:

1. **Improved Fault Tolerance**: If a peer storing a file goes offline, the system dynamically replicates the file to new peers to prevent data loss.

2. **Balanced Load Distribution**: When a file becomes popular and receives a large number of requests, the system automatically replicates the file to additional peers, reducing the load on the original host peer.

4. Approach and Plan

Components:

1. **Napster Core System:**
   We will first implement the core functionality of the original **Napster architecture**, which operates on a **centralized index server**. This index server keeps track of files shared by peers (clients). The process includes:
   1. **Central Indexing**: The index server maintains a list of files available on various peers and the IP addresses of those peers. Each time a peer logs in, it registers its list of shared files with the index server.
   2. **File Search and Download**: When a peer wants to download a file, it queries the index server for peers hosting that file. The index server responds with a list of peers, and the peer can directly connect to one of those peers to download the file.
   3. **Direct Peer-to-Peer Transfer**: File transfers happen directly between peers, without involving the central server beyond the initial lookup.
2. **On-demand replication module**:
   - **Monitoring file requests**: Track the number of requests a file receives and trigger replication when a predefined threshold is reached (e.g., 10 download requests within a minute).
   - **Peer selection for replication**: Choose peers based on their availability, storage capacity, and current load. Peers with high uptime and lower bandwidth usage will be prioritized.
3. **Replication management**:
   - Implement logic for dynamically creating and managing file replicas. Files that are replicated on-demand should be tracked in a central or semi-centralized index to avoid unnecessary duplications.
   - Once a file becomes less popular, the number of replicas can be reduced or consolidated to free up resources.
4. **Fault tolerance and recovery**:
   - If a peer holding a replicated file goes offline, the system should detect the failure and reassign the replication responsibility to another active peer.
5. **Load balancing on the broker**:
   - Implement a **load balancer** that redirects incoming file requests to the least loaded peers. This prevents any single peer from becoming overwhelmed by traffic.

Technologies to be used:

- Use gRPC for broker-client communication.
- A message passing interface to communicate between clients (TBD)

Dataset:

We will use a sample dataset consisting of a variety of MP3 files (audio) to simulate real-world usage scenarios and test the dynamic replication feature under different loads.

5. Timeline

- **Week 1 (October 25–October 31)**: Set up basic P2P architecture and implement a simple file-sharing protocol. Design the central indexing system and build the replication tracking mechanism.
- **Week 2-3 (November 1–November 14)**: Implement the on-demand replication feature. Develop logic for monitoring file requests and dynamically replicating files based on demand.
- **Week 4 (November 14–November 21)**: Integrate the load balancing component to distribute file requests evenly across peers holding replicated files. Test the system with sample datasets to monitor performance.
- **Week 5 (November 22–November 28)**: Conduct extensive testing under different failure and load conditions. Optimize the system for scalability and performance. Prepare final project report and presentation.