RafBook - Concurrent and Distributed Systems

System Architecture

The system architecture enhances the Chord distributed system framework by integrating new functionalities for file management and robust operations. This design preserves the core principles of Chord while adding features to improve flexibility and reliability.

Distributed Fair Mutex

The distributed fair mutex in this system is based on the Suzuki-Kasami algorithm, adapted to fit the Chord architecture. The mechanism involves:

- **Single Token Management**: A single token allows access to the critical section. There is only one token in the whole system and whoever has got it has access to the CS.
- **REQUEST Messages**: Nodes without the token broadcast REQUEST messages.
- **Sequence Number List**: Each node maintains a list of sequence numbers for all other nodes, tracking the latest request from each.
- **Token Handling**: The token includes:
  - A queue for the next nodes to access the critical section.
  - A list of the last completed requests for each node.
- **Queue Processing**: The node holding the token checks for nodes with sequence numbers incrementing by one relative to the token and queues them for access.

Concurrency Management:
- Multiple threads within a node can request access to the critical section, but only one thread is allowed access at a time.
- In case of failure detection, the requesting thread gets immediate priority for the token.

Detailed explanation of the algorithm.

File Management

Adding a New File

To add a new file with a unique name and path:

1. **ValueMap Structure**: Each node's `ChordState` includes a distributed hash table called valueMap.
   - **Primary Map Key**: Derived from the hashed file path.
   - **Internal Map**: With the file path as the key and `MetaFile` (containing metadata like privacy status and ownership) as the value.
2. **Node Responsibility**: The system locates the node responsible for the file key.
3. **File Addition Process**:
   - Request the token.
   - Send a PUT message to the responsible node.
   - The node adds the file metadata to its `valueMap` and backs it up with successor and predecessor, meaning that our system can hold up against any two nodes failing simultaneously.
   - Notify that addition is done with a PUT_UNLOCK message to unlock the distributed mutex.

Adding a Node to the Friends List

Nodes can subscribe to each other, enabling access to private files:
- **One-way subscription**: Node A subscribing to Node B allows A to view B's private files, but not vice versa.

Fetching a File

To retrieve any file from the system:

1. **Hashing**: Hash the file path to locate the responsible node.
2. **Value Retrieval**: Check the `valueMap` of the responsible node and return the data.
3. **Privacy Check**: Ensure access rights based on the file's privacy settings.

4. **Process**:
   - Request the token.
   - Send an ASK_GET message to the responsible node.
   - Forward the message if necessary.
   - Return the data with a TELL_GET message after verifying access rights.

Removing a File

To remove a file from the network:

1. **Locate Responsible Node**: Identify the node responsible for the file key.
2. **Ownership Verification**: Ensure the requester is the file owner.
3. **Deletion Process**:
   - Send a DELETE message to adjacent nodes.
   - Forward the message using exponential jumps until it reaches the responsible node.
   - Delete the file and inform neighbors with a DELETE_BACKUP message.
   - Confirm with a DELETE_UNLOCK message.

Node Management

Proper Node Shutdown

For a node to shut down properly:

1.**Token Request**:The node requests the token from all its neighbors.
2. **SHUTDOWN Message**: Send a SHUTDOWN message to the successor.
3. **System Update**: The message propagates through the system, updating all nodes to remove the shutting-down node from their tables.

Fault Tolerance

The system ensures fault tolerance through regular checks and broadcasts:

1. **PING/PONG Mechanism**
   - Nodes periodically send PING messages to their predecessors.
   - If there is no PONG response within the WEAK_LIMIT, a broadcast request checks the node's status.
2. **Failure Detection**:
   - Reset the timer if contact is reestablished.
   - If no response beyond the STRONG_LIMIT, the node is considered dead.
   - Broadcast a removal message to all nodes, instructing them to eliminate the unresponsive node from the system.