P2P File-Sharing System in Golang - Design Document

Juan Pedrajas

Architecture

The architecture consists of a hierarchical peer-to-peer network with super-peers acting as proxies or brokers. Weak peer nodes connect exclusively to a super-peer and perform tasks such as hosting files, registering/unregistering with their associated super-peer, and updating file information dynamically. Super-peer nodes manage connections with peers and neighboring super-peers, facilitating operations such as peer registration, file listing, updates, and query propagation. When a peer searches for a file, a query message is sent to its super-peer, which broadcasts it to neighboring super-peers. The system employs time-to-live (TTL) values to prevent infinite message forwarding loops and maintains logs of all operations performed on super-peer nodes for monitoring and auditing purposes.

Implementation of the P2P

Week Nodes

The modified query protocol now includes additional parameters to accommodate the new functionalities. Each query message contains the following parameters:

1. **Message ID**: A globally unique identifier for the query message, allowing tracking and correlation with corresponding query hit messages.

2. **Sender ID**: The unique identifier of the sender node, enabling identification of the originator of the query message.

3. **Time to Live (TTL)**: A numerical value representing the maximum number of hops or intermediary nodes that the query message can traverse before it expires. The TTL is decremented at each hop, preventing indefinite propagation and ensuring efficient message delivery while avoiding network congestion and infinite loops.

These parameters enhance the query protocol by enabling reliable message routing, identification of message originators, and effective management of message propagation within the peer-to-peer network. With these additions, the query protocol supports the propagation of query messages through the network, facilitating file searches and retrieval while maintaining scalability and efficiency.’.

Super Nodes

1. **Broadcast Query Function**:
    - When a super-peer receives a query message from a peer node, it first checks if the TTL is greater than 0.
    - If the TTL is greater than 0, the super-peer decrements the TTL by 1 and broadcasts the query message to all its neighboring super-peers.
    - The super-peer also keeps track of the message ID and sender ID to avoid redundant query propagation.
    - If the TTL is 0 or the query has already been processed, the super-peer discards the query message to prevent unnecessary processing and network overhead.
2. **Recursive Querying**:
    - Upon receiving a query message, each neighboring super-peer repeats the process, decrementing the TTL, broadcasting the query to its neighbors, and tracking message IDs.
    - If a super-peer receives a query message with a TTL of 0 or from a sender it has already processed a query from, it discards the message.
    - Super-peers continue to propagate the query until the TTL reaches 0 or a query hit is obtained.

By implementing these mechanisms, the super-peer node facilitates efficient file search and retrieval within the peer-to-peer network by recursively querying neighboring super-peers.

Trade-offs

1. **Network Overhead**: Broadcasting query messages to all neighboring super-peers can lead to increased network overhead, especially in large networks with many nodes. Each query message consumes bandwidth and processing resources at every hop it traverses. Recursive querying further amplifies this overhead as the message propagates through multiple levels of the network.
2. **Message Duplication**: With recursive querying, there's a risk of message duplication as query messages may be received multiple times by the same super-peer or neighboring nodes. This redundancy can contribute to inefficient resource utilization and may require additional processing to eliminate duplicate messages.
3. **Scalability**: The scalability of the system may be affected by the overhead introduced by recursive querying. As the network grows larger, the number of nodes and messages increases, potentially leading to performance degradation and longer query response times. Balancing the tradeoff between network size and query efficiency is crucial for maintaining system scalability.
4. **Latency**: Recursive querying can increase query latency as the message may traverse multiple hops before reaching a node with the requested file or before the TTL expires. The depth of the network hierarchy and the TTL value directly influence the latency experienced by query messages. Higher TTL values can increase the likelihood of reaching distant nodes but may also prolong the query process.

Future work

1. **Node Discovery**: Implement mechanisms for node discovery to enable new nodes to join the network. This could involve bootstrapping nodes or utilizing existing nodes as entry points for newcomers. Nodes could broadcast their presence or query known nodes for information about the network.
2. **Cache Integration with Super-Peers**: Integrate the cache mechanism with super-peers to leverage their proximity to multiple peers and optimize data caching. Super-peers can maintain their own caches or coordinate caching decisions among neighboring super-peers to maximize cache hit rates and reduce redundant data fetching.
3. **Supernode Redundancy Configuration**: Configure each weak node to be capable of hosting multiple supernodes. This involves setting up the necessary infrastructure and resources on each weak node to support multiple supernodes concurrently.

Conclusion / Summary

This architecture features a hierarchical network structure comprising super-peers and weak nodes, with super-peers acting as intermediaries for various network operations. The document delves into the implementation details of the system, highlighting the modifications made to the query protocol to enhance its functionality and efficiency. Furthermore, it explores the roles and responsibilities of weak nodes and super-peers, detailing their respective functions in file hosting, query propagation, and network management.

The design document also discusses the trade-offs inherent in the system, such as network overhead, message duplication, scalability challenges, latency concerns, and the importance of maintaining message integrity and security. These considerations underscore the complexity of developing and maintaining a robust P2P file-sharing system capable of supporting large-scale distributed file sharing while mitigating potential drawbacks.

Looking ahead, the document proposes future avenues for improvement, including mechanisms for node discovery, integration of caching mechanisms with super-peers, and configurations to enhance supernode redundancy. These suggestions provide valuable insights into potential enhancements to further optimize system performance, scalability, and resilience.

Overall, the design document presents a comprehensive overview of the P2P file-sharing system's architecture, implementation, trade-offs, and future directions. It serves as a valuable resource for developers and researchers seeking to understand and contribute to the advancement of peer-to-peer networking technologies.