Project Report: Google File System (GFS) with Dynamic Replication

TEAM : 37
Archit Pethani(2022101098)
Neel Amrutia(2022101117)

1. Base GFS Implementation

Overview

The Google File System (GFS) is a scalable distributed file system developed to handle large-scale data storage across commodity hardware. It is designed to provide high throughput, fault tolerance, and scalability, making it ideal for large datasets and applications like data processing and analytics.

System Overview

1. **Master Server**:
    - Manages metadata, including file-to-chunk mappings and chunk-to-chunkserver mappings.
    - Assigns chunks to chunkservers and tracks chunk replicas.
2. **Chunk Server**:
    - Stores data in fixed-size chunks.
    - Performs read, write, and replication operations based on requests from the master server or clients.
3. **Client**:
    - Interfaces with the master server to read and write data.
    - Sends operations like upload, read, and write requests to the appropriate chunkserver.

Key Features

- **Chunk-based Storage**: Files are split into fixed-size chunks, each identified by a unique handle.
- **Replication**: Chunks are replicated across multiple chunkservers to ensure reliability.
- **Heartbeat Mechanism**: Chunkservers send periodic pings to the master server to confirm they are alive.

Implementation Highlights

- The **Master Server** manages chunk distribution and metadata storage using data structures like dictionaries for fast lookup.
- The **Chunk Server** handles storage operations, responding to client read and write requests while periodically pinging the master server.
- The **Client** manages file uploads, downloads, and updates by communicating with the master server for chunk locations and chunkservers for data operations.

2. Dynamic Replication

Problem Statement

Static replication strategies used in traditional GFS implementations cannot dynamically adjust to changes in load or chunk access patterns. This limitation can lead to hotspots where certain chunkservers are overloaded while others are underutilized.

Solution

The implementation introduces **dynamic replication**, which adjusts the number of replicas based on the access rate to chunks. The master server monitors chunk access counts and dynamically increases or decreases replicas based on preset thresholds.

Implementation Details

1. **Chunk Access Monitoring**:
    - Access counts for each chunk are tracked over fixed intervals (here, 15 seconds).
2. **Replica Management**:
    - **Increase Replicas**: When access exceeds an upper threshold, new replicas are created on underloaded chunkservers.
    - **Decrease Replicas**: When access falls below a lower threshold, excess replicas are deleted to save resources.
3. **Load Balancing**:
    - The master server assigns new replicas to chunkservers with the lowest load, ensuring even distribution.
    - Similarly when the number of replicas have to be decreased then chunk is removed from the chunkservers with higher load.

Example Logic:

- **Increase Replication**: If a chunk's access rate exceeds the upper limit, calculate how many new replicas are needed and distribute them across the least loaded chunkservers.
- **Decrease Replication**: If a chunk's access rate drops below the lower limit, reduce the number of replicas while maintaining the minimum required replication.

Limitations

- **Delayed Adjustments**: Replica adjustments are based on periodic checks, which may introduce lag in response to rapid changes in load.
- **Minimum Replication Constraint**: A hard limit on the minimum number of replicas (e.g., 2) ensures fault tolerance but may hinder optimal resource usage.
- **Dependency on Load Metrics**: Inaccurate load metrics could lead to suboptimal replication decisions.

3. Conclusion and Future Work

Conclusion

The project successfully implements a GFS replica enhanced with dynamic replication to address load balancing and fault tolerance. It dynamically adjusts replication levels to optimize performance based on chunk access patterns, ensuring an even distribution of workload across chunkservers.

Future Work

1. **Enhanced Monitoring**: Implement finer-grained monitoring to reduce lag in detecting access pattern changes.
2. **Predictive Scaling**: Use machine learning to predict future access patterns and proactively adjust replication levels.
3. **Improved Fault Detection**: Enhance the failure detection mechanism to more quickly identify and recover from chunkserver failures.