Reflection Report

1. RPC and Message Passing

Question: What are the benefits of combining RPC with message passing in this project? How does message passing enhance the scalability of the server?

Answer:
Combining **Remote Procedure Calls (RPC)** with **message passing** creates a flexible and efficient architecture.

- **RPC Benefits**: RPC facilitates direct communication between the `paperclient` and `paperserver`, enabling synchronous and efficient operations such as adding, listing, and retrieving papers.
- **Message Passing via RabbitMQ**: Decouples the notification mechanism from core server operations. For example, when a new paper is added, a message is sent to RabbitMQ, allowing other services to receive updates asynchronously. This approach improves scalability by distributing workloads and reducing direct dependencies on the server.

2. Concurrency and Synchronization

Question: How does Go handle concurrent connections in the server? Why is it important to handle synchronization when using message passing?

Answer:

- **Concurrency Handling**: Go utilizes lightweight goroutines to manage concurrent client connections efficiently. This allows the `paperserver` to process multiple requests simultaneously without blocking other clients, ensuring high performance under load.
- **Synchronization Importance**: Shared resources like the in-memory paper store must be protected from race conditions. A `sync.Mutex` is used to lock the resource during critical operations, ensuring data integrity and preventing inconsistent states during concurrent accesses.

3. Reliability and Fault Tolerance

Question: What would happen if the RabbitMQ service went down? How could you make the notification service more resilient?

Answer:

- **RabbitMQ Downtime**: If RabbitMQ becomes unavailable, the notification mechanism would temporarily fail, but the core functionalities of the `paperserver`—such as adding, listing, and retrieving papers—would remain unaffected. This separation ensures the system's core services continue operating independently.

- **Enhancing Resilience**: Implementing RabbitMQ message persistence ensures that notifications are stored on disk and delivered once RabbitMQ is back online. Additionally, configuring RabbitMQ in a clustered or high-availability setup would further improve reliability.

4. File Storage in Memory

Question: What are the limitations of storing paper content in memory, and how would you modify this design for a larger system?

Answer:

- **Limitations**: In-memory storage is fast and simple but not scalable for large datasets. It is also volatile, meaning all stored data is lost if the server restarts or crashes.
- **Improvements for Larger Systems**: Transitioning to a persistent database like PostgreSQL or MongoDB would address scalability and durability concerns. This would allow efficient querying, support for large data sets, and ensure data is not lost in case of server failures.

5. Real-World Applications

Question: How could the system be extended for more features, such as keyword search or paper downloads by multiple formats?

Answer:
To make the system more versatile and applicable in real-world scenarios, the following features can be added:

- **Keyword Search**: Integrating a search index, such as Elasticsearch, would allow users to search papers by title, author, or keywords efficiently.
- **Multi-Format Downloads**: Supporting various paper formats (e.g., PDF, DOCX) would increase accessibility. Conversion tools like LibreOffice or online APIs can be used for format conversion.
- **Distributed Storage**: Cloud-based storage solutions such as AWS S3 or Google Cloud Storage would enable better scalability, availability, and global access to stored papers.