Escalabilidad:

- Preocupación (Concern):¿Cómo manejar un número creciente de solicitudes sin degradar la calidad del servicio?

- Táctica: Usar un balanceador de carga para distribuir las solicitudes entrantes entre varios servidores. En este caso, se ha implementado un balanceador de carga a nivel de aplicación.

- Patrón de Arquitectura: Se ha aplicado el patrón de arquitectura de balanceo de carga (Load Balancing). Este patrón permite que el sistema maneje más solicitudes simplemente añadiendo más servidores.

Desempeño:

- Preocupación (Concern): ¿Cómo asegurarse de que el sistema responde rápidamente a las solicitudes de los usuarios?

- Táctica: Uso de multithreading en el servidor para manejar varias solicitudes simultáneamente. Cada hilo maneja una solicitud, permitiendo que el sistema maneje múltiples solicitudes a la vez sin bloquearse.

- Patrón de Arquitectura: Se ha utilizado el patrón de arquitectura basada en eventos (Event-Driven Architecture). En particular, se está usando el modelo de Reactor, donde los hilos del servidor manejan las solicitudes cuando están listas.

3. **Modificabilidad**:

- Preocupación (Concern): ¿Cómo facilitar futuros cambios en el código, como añadir nuevas características o cambiar las existentes?

- Táctica: Uso de interfaces y clases abstractas para definir comportamientos que pueden cambiar en el futuro. Por ejemplo, `IServerScheduler` puede ser implementado de diferentes maneras para cambiar la estrategia de balanceo de carga.

- Patrón de Arquitectura: Se ha aplicado el patrón de Estrategia (Strategy Pattern). Este patrón permite cambiar el comportamiento de una clase en tiempo de ejecución al delegar esas responsabilidades a diferentes objetos que implementan una interfaz común.