PDC Assignment Report: API Communication, Microservices & Performance Comparison

This assignment focuses on implementing and comparing three modern API communication technologies: REST, tRPC, and gRPC. The objective is to analyze their performance in terms of latency, payload size, serialization method, and suitability for scalable systems. Additionally, gRPC-based service-to-service communication is implemented to demonstrate microservices efficiency.

1. REST API Implementation

REST (Representational State Transfer) is a widely used architectural style that uses HTTP methods and JSON for data exchange. In this implementation, a REST server was built using Node.js and Express. A client sends an HTTP GET request to retrieve user data. Latency was measured using timestamps before and after the request, while payload size was calculated using JSON string length.

Advantages: Simple, readable, widely supported.
Disadvantages: Larger payload size, higher latency due to JSON parsing.

2. tRPC API Implementation

tRPC is a type-safe API framework that removes the need for traditional REST endpoints. The tRPC server and client were implemented using Node.js. Unlike REST, the client directly calls server procedures, reducing boilerplate code and enabling automatic request batching. Performance measurements show reduced latency compared to REST while still using JSON.

Advantages: End-to-end type safety, reduced API overhead.
Disadvantages: Limited to TypeScript ecosystem, still JSON-based.

3. gRPC API with Protocol Buffers

gRPC uses HTTP/2 and Protocol Buffers for communication. In this task, a gRPC server and client were implemented using Node.js. Protobuf provides binary serialization, resulting in smaller payload sizes and lower latency compared to JSON-based approaches. This makes gRPC suitable for high-performance systems.

Advantages: Fast, efficient, strongly typed.
Disadvantages: Less human-readable, more complex setup.

4. gRPC Service-to-Service Communication

Two microservices were created: Service A provides user data, while Service B acts as an intermediary that receives client requests and communicates with Service A using gRPC. This architecture demonstrates how gRPC reduces internal latency and improves performance in microservices-based systems.

Conclusion

The results show that REST is best suited for simple and public APIs, tRPC is ideal for frontend-backend communication with type safety, and gRPC provides the best performance for microservices and large-scale distributed systems. Overall, gRPC offers the lowest latency and smallest payload size, making it the preferred choice for internal service communication.