Professional paper

USING THE GRPC PROTOCOL IN THE CONTEXT OF CRISIS MANAGEMENT

Lozić, D.[1], Bralić, V.[2], Filipović, A. M.[3]

[1] Zagreb, dlozic@vvg.hr
[2]University of Applied Sciences Velika Gorica,
Velika Gorica, vladimir.bralic@vvg.hr
[3]Croatian Academic and Research Network – CARNET,
Zagreb, antun.matija.filipovic@carnet.hr

Abstract: *Speed, stability, and efficiency are key parameters in inter-service communication for crisis management. Traditionally, microservice architecture has utilized REST (Representational State Transfer), an architectural style based on the principles of the HTTP protocol, due to its simplicity and broad support. However, with the increasing number of requests to and between microservices, the gRPC protocol is becoming an increasingly attractive alternative. This paper presents the technical advantages of the gRPC protocol over traditional REST in the context of inter-service communication for crisis management, focusing on several key parameters: low latency, efficiency, data streaming capabilities, and support for distributed transactions. The paper introduces the gRPC protocol through the Go programming language, which is also recognized as a highly efficient, reliable, and scalable language, making it very suitable for building microservices.*

Keywords: *gRPC, REST, streaming, security*

INTRODUCTION

In crisis situations, where speed and reliability of communication are crucial, the choice of technology for data transfer can significantly impact the effectiveness of the solution. Traditionally, many systems use REST (Representational State Transfer) for communication between different services and applications. While REST offers a simple and widely accepted way of exchanging data, there are alternative technologies that can provide significant advantages in the context of crisis situations. One such technology is gRPC (gRPC Remote Procedure Call).

gRPC is a modern system for remote procedure calls (RPC) that enables fast, efficient, and secure communication between distributed applications and services. This protocol, developed by Google, uses Protobuf (Protocol Buffers) for data serialization and is based on the HTTP/2 protocol. These characteristics make gRPC particularly suitable for environments where performance, reliability, and security are of utmost importance.

One of the most significant advantages of gRPC is its efficiency. Thanks to the use of the binary Protobuf protocol, gRPC allows for much more compact and faster data transfer compared to JSON, which REST uses. Smaller message sizes reduce network load and speed up communication, which is extremely important in situations where every millisecond counts. Additionally, Protobuf enables efficient data compression, further reducing network traffic and accelerating information transfer.

The use of the HTTP/2 protocol brings additional benefits. HTTP/2 allows for multiplexing, meaning multiple requests can be sent over a single TCP connection without waiting. This significantly reduces latency and increases the speed of communication.

Furthermore, gRPC supports bidirectional streaming, enabling the client and server to exchange messages simultaneously. This functionality is particularly useful for applications that require continuous two-way communication, such as systems for monitoring and managing natural disasters.
Reliability and stability are also key features of gRPC. Providing advanced error handling mechanisms enables quick detection and resolution of issues, which is vital in crisis situations.
Simplicity and consistency of implementation are additional advantages brought by gRPC. Defining services and methods through .proto files allows for straightforward and consistent implementation on both the client and server sides. Automatic code generation from these files reduces the possibility of errors and speeds up development, which is especially important in crisis situations where time is a critical factor.
Scalability is another advantage of gRPC. This protocol is designed to work in distributed systems and can easily scale horizontally which is important for solutions used in large-scale crisis situations. Support for advanced load balancing allows for efficient resource management and request distribution ensuring that the system can handle a large number of simultaneous requests without overload.
Security is yet another aspect where gRPC excels. Built-in support for authentication and TLS encryption ensures secure data transmission which is essential in crisis situations where data is often sensitive and confidential.
Practical examples of gRPC application in crisis situations include systems for monitoring and managing natural disasters where fast and reliable communication for real-time data transfer is needed. Additionally, gRPC allows for easier integration with different systems and devices, which is often the case in complex crisis situations involving various organizations and technologies.

PROTOCOL OVERVIEW

The evolution of web communication protocols and architectures has significantly shaped the modern internet landscape. HTTP, since its standardization, has been fundamental to the World Wide Web, evolving from HTTP/1's basic functionality to HTTP/3's advanced capabilities over the QUIC protocol. REST, an architectural style for web services, leverages HTTP for client-server communication with stateless interactions and standardized methods. Meanwhile, gRPC, developed by Google, represents a high-performance alternative, utilizing HTTP/2 and Protocol Buffers for efficient and scalable inter-service communication, making it ideal for complex microservices environments.

1.1. HTTP

Since its standardization, HTTP became the backbone of the World Wide Web. It's a protocol made for distributing hypermedia and since it is an application level protocol it is agnostic to, otherwise important, network aspects like transmission and addressing. (Gourley & Totty, 2002) There were big improvements from HTTP/1 to today's current version HTTP/3.
HTTP/1.1 introduced persistent connections as the earlier version had to open new TCP connection for every request. It introduced chunked transfers of data, additional caching mechanisms and more specific status code. (Fielding, et al., 1999)
HTTP/2 was a major revision which allowed multiple requests to be processed non-sequentially meaning that multiple requests and responses could be interleaved over a single connection. It implemented header compression, reducing the overhead as a consequence and it allowed server push which is a major improvement since requests on earlier versions had to be initiated exclusively from a client. (Belshe, Peon, & Thomson, 2015)
Latest version, HTTP/3 (Thomson & Benfield, 2021) allows communication over QUIC (Quick UDP Internet Connections) protocol. This significantly reduces the latency as UDP doesn't require an extensive handshake like the TCP. QUIC supports connection migration which allows clients to have a

persistent connection even when changing networks, like changing between Wi-Fi and cellular. (Iyengar & Thomson, 2021)

1.2. REST

REST (Representational State Transfer) is an architectural style used for creating web/network services. It leverages HTTP to enable client-server communication. Primary concept is called resource which can be any document, image, collection, etc. Each resource is identified by the Uniform Resource Identifier (URI). (Fielding R. T., 2000)
REST uses standardized interface by leveraging HTTP's GET, POST, PUT, PATCH, DELETE operations. This makes the requests more stable and predictable, for example, GET is used when retrieving the data, POST is used when sending the data. (Richardson & Ruby, 2008.)
Each request from client to the server must contain all information needed for understanding the request and each request is independent of the previous one, which makes it stateless: the server doesn't store any information about the client.
While REST over HTTP could be used for inter-service communication, newer players on the block came to solve important issues: performance, latency, streaming and better error handling.

1.3. gRPC

gRPC (gRPC Remote Procedure Calls) is a high performance RPC framework developed by Google to enable scalable and efficient communication between services. Built on HTTP/2 is a big leap over REST-based communication. It's suitable for high-performance environments such as microservices architectures. (Indrasiri & Kuruppu, 2020)
gRPC utilizes a language-agnostic binary serialization tool, Protocol Buffers (Protobuf). It not only enforces a structured contract (API) between client and server but ensures efficient data encoding. (Indrasiri & Kuruppu, 2020)
It supports several service methods: unary RPCs, client streaming, server streaming and bidirectional streaming.

PROTOCOL EFFICIENCY COMPARISON: GRPC VS. REST

When it comes to the efficiency of communication protocols, gRPC and REST have different approaches, particularly in how they handle connection management and data serialization. These differences are crucial for applications speed and performance.
gRPC utilizes Protocol Buffers (Protobuf), a language-neutral and platform-neutral mechanism for serializing structured data. Unlike JSON, which is a text-based format, Protobuf is a binary format that is significantly more efficient in terms of speed and size. This efficiency comes from the way Protobuf encodes data into a compact binary format, reducing the amount of data that needs to be transmitted over the network. (Google LLC, 2024)
Consider the following example where data is about an earthquake event. Using JSON, the data might look like this:

```
{
  "event": "earthquake",
  "location": {
    "latitude": 45.8008,
    "longitude": 15.99
  },
  "magnitude": 6.5,
  "timestamp": "2024-20-06T10:00:05Z"
```

```
}
```

In gRPC Protobuf same data would be defined in a `.proto` file and then serialized, resulting in a much more compact representation. The `.proto` file might look like this:

```
message Earthquake {
    string event = 1;
    Location location = 2;
    float magnitude = 3;
    string timestamp = 4;
}

message Location {
    double latitude = 1;
    double longitude = 2;
}
```

Protobuf message is smaller compared to its JSON counterpart. The binary encoding of Protobuf reduces the payload size significantly, leading to faster transmission time and lower bandwidth usage. This is critical in mobile software or services operating in areas with limited connectivity where network performance. (Ramu, 2023)

The improved network performance of the gRPC compared to REST has been well studied. A recent study (Niswar, Safruddin, Bustamin, & Aswad, 2024) presented benchmark data showing gRPC to have improved performance, compared to REST, in fetching data and CPU utilization. This study shows a response time of 4,009.83 ms for REST as opposed to 2,606.59 ms for gRPC when fetching 500 requests for flat data. Fetching nested data yielded much tighter results with rest taking 16,646.55 ms and gRPC taking 14,926.61 ms for 500 requests. When it comes to CPU utilization the tests performed by Niswar et al. once again show an advantage for gRPC, with gRPC scoring a 30.11% CPU utilization and REST scoring a 38.23% utilization for 100 requests of flat data. Nested data benchmarks were almost identical, with gRPC scoring a 10.95 % and REST a 10.26% utilization for 100 requests. (Niswar, Safruddin, Bustamin, & Aswad, 2024)

Payload size and lower bandwidth usage are not the only factors important in critical situations. Ability to stream data and allow continuous flow is also crucial for monitoring and alerting in such scenarios.

DATA STREAMING

gRPC supports client streaming, server streaming and bidirectional streaming all of which have an important usage in critical situations. (Indrasiri & Kuruppu, 2020)

1.4. Client Streaming

Client streaming allows a client to get back a response once a client initiates a data stream to the server. This is useful when continuous collection from multiple data source needs to be analyzed and aggregated by a central system. (Grigorik, 2013)

In the case of, for example, an earthquake, sensors deployed across multiple locations can continuously stream seismic data to a central server. The server can aggregate collected data and analyze it in real-time. The data is transmitted as it is created ensuring that the server has the most up-to-date data without waiting for a complete dataset.

1.5. Server Streaming

gRPC also supports server streaming which allows a server to send a stream of responses to a client after the client sends a single request. This option is useful when the client needs to receive continuous updates from the server in response to a single query. (Grigorik, 2013)
In an earthquake scenario, an emergency response application could send a request for updates to a central server. The server can then continuously stream information back to the application, such as updated damage assessment, aftershock warnings, and instructions for evacuation. This continuous flow of information ensures that responders and the public are kept informed.

1.6. Bidirectional Streaming

Bidirectional streaming enables both the client and server to send a stream of messages to each other independently. For instance, in the event of an earthquake, bidirectional streaming can enable dynamic interaction between command centers and field units. Rescue teams with mobile devices can stream live video, GPS coordinates, and status updates to a central command center. At the same time, the command center can provide critical information such as new rescue priorities, safe routes, and real-time situation analysis. This continuous, two-way communication ensures synchronization between the command center and field units. (Grigorik, 2013)

DISTRIBUTED TRANSACTIONS

Distributed systems often involve various services that must work together to manage the complexity and urgency of the situation. (Bellemare, 2020) For instance, during an earthquake, services such as emergency alerting, data collection, rescue coordination and resource allocation need to be tightly integrated.
gRPC, with its support for efficient and real-time communication has an important role in enabling this coordination. By allowing different services to communicate rapidly, gRPC ensures that each component of the emergency response system can share data and updates without delay. This capability is vital when managing distributed transactions where multiple operations must be coordinated across various services to ensure a consistent and accurate outcome. (Indrasiri & Kuruppu, 2020)
Transactions by their nature involve multiple networked resources or services that should be managed in a way that they all succeed or fail together. In the context of earthquakes, there are scenarios that should maintain overall system integrity in order to have a correct information on time. For example, resource allocation should simultaneously allocate rescue teams depending on seismic data collection based on real-time data.
In this scenario, distributed transactions ensure that if any part of the process fails (e.g., a sensor goes offline, or a resource allocation service encounters an error), the system can roll back or compensate to maintain a consistent state. This prevents partial updates that could lead to misinformation or inefficient resource deployment.

DISTRIBUTED TRANSACTION IMPLEMENTATION WITH GRPC

gRPC manages distributed transactions through its support for high-performance, bidirectional streaming, and its ability to maintain persistent connections. As for all transactions, it is important they are atomic, consistent, isolated, and durable. (Kleppmann, 2017)

1.7. Isolation

Each service involved in the transaction can operate independently but also remain in-sync through gRPC's streaming capabilities. For example, alert system can independently verify seismic data while

the resource allocation service prepares for possible deployment. All that is happening while the overall transaction is isolated until it is ready to commit.

1.8. Consistency and atomicity

Services can use gRPC to communicate status updates and transaction states in real-time. For example, a seismic data service streams continuous updates to the alerting service. If a critical threshold is detected the alerting service starts the transaction by notifying other connected services.

1.9. Durability

gRPC ensures that once a transaction is committed, the data stays consistent across all services. For example, gRPC guarantees that all services maintain same status after an alert is sent and rescue resources are deployed.

CONCLUSION

gRPC offers significant advantages over REST in crisis management, such as increased speed, efficiency, and reliability. Using Protocol Buffers and HTTP/2, gRPC ensures low-latency, efficient data transmission and robust bidirectional streaming, crucial for real-time data processing in emergencies. It reduces network load and speeds up communication through binary serialization, while multiplexing decreases latency for timely data exchange. gRPC's support for distributed transactions and scalability ensures consistent operations under high demand. Practical applications like real-time seismic data monitoring and communication between command centers and field units highlight its potential to enhance emergency response effectiveness and resilience. gRPC's technical strengths make it ideal for crisis management, improving the effectiveness and resilience of emergency response solutions.

Further insight into the details and study of the protocol should include other, well cited, papers such the work by Wang, Zhao and Zhu describing the (Wang, Zhao, & Zhu, 1993) and books on the subject such as “gRPC: up and running: building cloud native applications with Go and Java for Docker and Kubernetes” (Indrasiri & Kuruppu, 2020).

REFERENCES

Bellemare, A. (2020). *Building event-driven microservices.* O'Reilly Media, Inc.

Belshe, M., Peon, R., & Thomson, M. (2015). *RFC 7540: Hypertext Transfer Protocol Version 2 (HTTP/2).* Retrieved from Internet Engineering Task Force: https://www.rfc-editor.org/info/rfc7540

Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures. Irvine: University of California.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, R., & Berners-Lee, T. (1999). *RFC 2616: Hypertext Transfer Protocol -- HTTP/1.1.* Retrieved from Internet Engineering Task Force: https://www.ietf.org/rfc/rfc2616.txt

Google LLC. (2024). *Protocol Buffers*. Retrieved from Protocol Buffers Documentation: https://protobuf.dev/

Gourley, D., & Totty, B. (2002). *HTTP: The Definitive Guide.* O'Reilly Media.

Grigorik, I. (2013). *High Performance Browser Networking.* O'Reilly Media, Inc.

Indrasiri, K., & Kuruppu, D. (2020). *gRPC: up and running: building cloud native applications with Go and Java for Docker and Kubernetes.* O'Reilly Media, Inc.

Iyengar, J., & Thomson, M. (2021). *RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport.* Retrieved from Internet Engineering Task Force: https://www.rfc-editor.org/info/rfc9000

Kleppmann, M. (2017). *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems.* O'Reilly Media, Inc.

Nielsen, H. F., Gettys, J., Baird-Smith, A., Prud'hommeaux, E., Lie, H. W., & Lilley, C. (1997). Network performance effects of HTTP/1.1, CSS1, and PNG. *Proceedings of the ACM SIGCOMM'97 conference on Applications, technologies, architectures, and protocols for computer communication*, 155-166.

Niswar, M., Safruddin, R. A., Bustamin, A., & Aswad, I. (2024). Performance evaluation of microservices communication with REST, GraphQL, and gRPC. *International Journal of Electronics and Telecommunication, 70*(2), 429-436. Retrieved 1 25, 2025, from https://ijet.pl/index.php/ijet/article/view/10.24425-ijet.2024.149562/2855

Ramu, V. B. (2023). Performance Impact of Microservices Architecture. *The Review of Contemporary Scientific and Academic Studies, 3*(6). Retrieved from https://thercsas.com/wp-content/uploads/2023/06/rcsas3062023010.pdf

Richardson, L., & Ruby, S. (2008.). *RESTful web services.* O'Reilly Media, Inc.

Thomson, M., & Benfield, &. (2021). *RFC 9114: HTTP/3.* Retrieved from Internet Engineering Task Force: https://www.rfc-editor.org/info/rfc9114

Wang, X., Zhao, H., & Zhu, J. (1993). GRPC: A communication cooperation mechanism in distributed systems. *ACM SIGOPS Operating Systems Review, 27*(3), 75-86.

KORIŠTENJE GRPC PROTOKOLA U KONTEKSTU UPRAVLJANJA U KRIZNIM UVJETIMA

Sažetak: *Brzina, stabilnost i efikasnost su ključni parametri u međuservisnoj komunikaciji pri upravljanju u kriznim situacijama. Tradicionalno se u mikroservisnoj arhitekturi koristio REST (engl. Representational State Transfer), arhitekturalni stil zasnovan na principima HTTP protokola – zbog svoje jednostavnosti i široke podrške. Međutim, povećanjem broja zahtjeva prema i između mikroservisa, protokol gRPC postaje sve privlačnija alternativa. Ovaj rad predstavlja tehničke prednosti gRPC protokola u odnosu na tradicionalni REST u kontekstu međuservisne komunikacije pri upravljanju u kriznim uvjetima, fokusirajući se na nekoliko ključnih parametara: nisku latenciju, efikasnost, mogućnost strujanja podataka (engl. Streaming) te podršku za distribuirane transakcije. Rad predstavlja protokol gRPC kroz Go programski jezik koji je također prepoznat kao vrlo efikasan, pouzdan i skalabilan jezik što ga čini vrlo prihvatljivim u izradi mikroservisa.*

Ključne riječi: *gRPC, REST, streaming, sigurnost*