***Review-1***

**File Sharing using gRPC**

**Subject:-** ***Operating Systems***

**Subject Teacher:- *Shaik Naseera***

**Slot:- *A2***

**Group Members:-**

|  |  |
| --- | --- |
| **Name** | **Registration Number** |
| *Saunak Sahoo* | 16BCE2167 |
| *Corrina Marieanne Barnabas* | 16BCE0754 |
| *Anurag B.G.* | 16BCE2298 |
| *Shaik Muzammil* | 16BCE0481 |

**INTRODUCTION**

The **RPC** (remote procedure call) architecture is popular in building scalable distributed client/server on a different address space without understanding the network configuring as if the server was in the same network. It is very simple and transparent to write distributed programs. The **RPC** protocol user need not to have information on distributed environments and can easily construct distributed application systems. The **RPC** protocol reduces the communications overload since it uses messages to communicate.

Current **RPC** services implement a variety of semantics, with many of the differences related to how communication and server failures are handled. The list increases even more when considering group **RPC**, a variant of **RPC** often used for fault-tolerance where an invocation is sent to a group of servers rather than one.

**RPC** messages:

- **Call/Request** is given by the Client to the Server.

- **Reply** is given back to the Client by the Server.

**gRPC** is a modern open source high performance **RPC** framework that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication. It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services.

As mentioned, with **gRPC**, a client application can directly call methods on a server application on a different network as if the method was local. The beauty about **RPC** is that it is language agnostic. This means we could have a **grpc** server written in **Java** handling client calls from **node.js, PHP** and **Go**

**gRPC** allows us to define a service which specifies the methods that you can call remotely, the parameters that can be passed to the procedure call and the return responses. The server then implements these definitions and creates a new **grpc** server to handle the procedure calls from the client.

**Topic Details**

**gRPC** is a Remote Procedure Calls tool built by Google. It is an open source platform or also an RPC framework which has been in operation since 2015. Essentially it allows for service to service communication. It seeks to make the process of creating libraries and APIs easier for developers. Developers now don’t have to write their libraries in different languages, instead they can do so in a unified manner and share their services with others without having to worry about translating it into other languages.

**Plan of action:**

* Learn about what gRPC is and how is it used.
* Start developing basic functions in gRPC.
* After gaining confidence in those, start using various other concepts of RPCs and implement them.
* Try using our own functions for service to service communications.

**ABSTRACT**

**Remote Procedure Call** (**RPC**) is basically a protocol that one program can use to request a service from a program located in another computer on a network without having to understand the network's details.

**gRPC** is a modern open source high performance RPC framework that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication. It is also applicable in last mile of distributed computing to connect devices, mobile applications and browsers to backend services.

As mentioned, with gRPC, a client application can directly call methods on a server application on a different network as if the method was local. The beauty about RPC is that it is language agnostic. This means we could have a grpc server written in Java handling client calls from node.js, PHP and Go.

RPC client and server run-time stubs take care of the network protocol and communication so that we can focus on our application.

We would be testing the functioning of the RPC, we would also be creating and testing the server including the gRPC server, we will also be creating gRPC clients.

**Literature Review**

**Design, implementation and performance evaluation of GridRPC programming middleware for a large-scale computational grid[1]: Tanaka, Optimization Problem Solving System using GridRPC by Hisashi Shimosaka[2]. Constructing a configurable group RPC service talks about Current Remote Procedure Call (RPC) services implementing a variety of semantics, with many of the differences related to how communication and server failures handling by Hiltunen[3]. A group RPC protocol for distributed systems by Lee[4]. OpenGR: A Directive-Based Grid Programming Environment by Hirano[5]. Design and implementation of YANG based wireless access point controller using RPC by An, G., & Lee[6]. A Grid RPC Model and API for Advanced and Middleware Applications by Casanova[7]. A group RPC protocol analysis for distributed network management systems using Petri nets by Lee, J. K., Lee[8]. Standardized data management in GridRPC environments by Caniou, Y., Caron[9]. First test of a power-pulsed electronics system on a GRPC detector in a 3-Tesla magnetic field by Caponetto [10].**

**References**

**[1] Tanaka, Y., Takemiya, H., Nakada, N., & Sekiguchi, S. (2004, November). Design, implementation and performance evaluation of GridRPC programming middleware for a large-scale computational Grid. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on* (pp. 298-305). IEEE.**

**[2] Shimosaka, H., Hiroyasu, T., Miki, M., & Dongarra, J. (2005). Optimization Problem Solving System using GridRPC. *IEEE Transactions on Parallel and Distributed Systems*, *1*(1), 1.**

**[3] Hiltunen, M. A., & Schlichting, R. D. (1995, May). Constructing a configurable group RPC service. In *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on* (pp. 288-295). IEEE.**

**[4] Lee, C. S., Lee, K. H., & Lee, J. K. (1997, September). A group RPC protocol for distributed systems. In *Information, Communications and Signal Processing, 1997. ICICS., Proceedings of 1997 International Conference on* (Vol. 2, pp. 805-809). IEEE.**

**[5] Hirano, M., Sato, M., & Tanaka, Y. (2003, October). OpenGR: a directive-based grid programming environment. In *International Symposium on High Performance Computing* (pp. 552-563). Springer, Berlin, Heidelberg.**

**[6] An, G., & Lee, S. (2017, July). Design and implementation of YANG based wireless access point controller using RPC. In *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on* (pp. 601-603). IEEE.**

**[7] Casanova, H., & UCSD, S. (2004). A GridRPC Model and API for Advanced and Middleware Applications.**

**[8] Lee, J. K., Lee, C. S., & Lee, K. H. (1997, October). A group RPC protocol analysis for distributed network management systems using Petri nets. In *Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on* (Vol. 5, pp. 4382-4387). IEEE.**

**[9] Caniou, Y., Caron, E., Le Mahec, G., & Nakada, H. (2011, November). Standardized data management in GridRPC environments. In *Computer Sciences and Convergence Information Technology (ICCIT), 2011 6th International Conference on* (pp. 501-508). IEEE.**

**[10] Caponetto, L., Combaret, C., De La Taille, C., Dulucq, F., Kieffer, R., Laktineh, I., ... & Seguin-Moreau, N. (2012). First test of a power-pulsed electronics system on a GRPC detector in a 3-Tesla magnetic field. *Journal of Instrumentation*, *7*(04), P04009.**