Remote Procedure Call as a Managed System Service

Jingrong Chen*, Yongji Wu*, Shihan Lin, Yechen Xu, Xinhao Kong, Thomas Anderson, Matthew Lentz, Xiaowei Yang, Danyang Zhuo

*Equal contributions
Remote Procedure Calls Widely Used

- etcd: Distributed Data Store
- GlusterFS: Network Filesystem
- Apache Spark: Data Analytics Framework
- Kubernetes: Cluster Orchestrator
- raft: Consensus Protocol
- TensorFlow: Deep Learning System
- PyTorch: Deep Learning System
Remote Procedure Calls Widely Used

In Google’s datacenter, RPCs

- generate >95% of application traffic[1]
- spend ~10% of its CPU cycles[2]

[1] Aequitas: Admission Control for Performance-Critical RPCs in Datacenters, SIGCOMM ’22
[2] Profiling a Warehouse-Scale Computer, ISCA ‘15
Remote Procedure Calls Widely Used

In Google’s datacenter, RPCs
- generate >95% of application traffic[1]
- spend ~10% of its CPU cycles[2]

Performance is always a key design goal of RPC

[1] Aequitas: Admission Control for Performance-Critical RPCs in Datacenters, SIGCOMM ’22
[2] Profiling a Warehouse-Scale Computer, ISCA ‘15
Remote Procedure Calls: Implementation
Remote Procedure Calls: Implementation

1. Write protocol specification

Service KVStore
Message GetReq
  bytes key
Message Entry
  bytes? value
Func Get(GetReq) -> Entry
Remote Procedure Calls: Implementation

1. Write protocol specification

Service KVStore
Message GetReq
  bytes key
Message Entry
  bytes? value
Func Get(GetReq) \rightarrow Entry
Remote Procedure Calls: Implementation

1. Write protocol specification

   Service KVStore
   Message GetReq
     bytes key
   Message Entry
     bytes? value
   Func Get(GetReq) -> Entry

2. Protocol compiler generates stub code

3. App compiles with the stub and RPC library
Remote Procedure Calls: Implementation

1. Write protocol specification
   
   **Service** KVStore
   **Message** GetReq
   bytes key
   **Message** Entry
   bytes? value
   **Func** Get(GetReq) --> Entry

2. Protocol compiler generates stub code

3. App compiles with the stub and RPC library

Remote Procedure Calls: Implementation

1. Write protocol specification

```
Service KVStore
  Message GetReq
    bytes key
  Message Entry
    bytes? value
  Func Get(GetReq) -> Entry
```

2. Protocol compiler generates stub code

3. App compiles with the stub and RPC library


RPC-as-a-library
- gRPC, Thrift, eRPC
- Cap’n Proto, rpclib, XML-RPC
- brpc, tarpc, tonic…
The Need for Manageability
The Need for Manageability

Observability

e.g., How many RPCs? RPC Latency?
The Need for Manageability

Observability
  e.g., How many RPCs? RPC Latency?

Policy Enforcement
  e.g., Prioritize certain RPCs?
The Need for Manageability

**Observability**
e.g., How many RPCs? RPC Latency?

**Policy Enforcement**
e.g., Prioritize certain RPCs?

**Upgradability**
e.g., Fix vulnerabilities while app running?
The Need for Manageability

- **Observability**
  - e.g., How many RPCs? RPC Latency?

- **Policy Enforcement**
  - e.g., Prioritize certain RPCs?

- **Upgradability**
  - e.g., Fix vulnerabilities while app running?
Manageability in a feature-rich RPC library?

Observability
e.g., How many RPCs? RPC Latency?

Policy Enforcement
e.g., Prioritize certain RPCs?

Upgradability
e.g., Fix vulnerabilities while app running?
Manageability in a feature-rich RPC library?

Observability
  e.g., How many RPCs? RPC Latency?  ➔ YES

Policy Enforcement
  e.g., Prioritize certain RPCs?

Upgradability
  e.g., Fix vulnerabilities while app running?
Manageability in a feature-rich RPC library?

**Observability**
- e.g., How many RPCs? RPC Latency? ➔ **YES**

**Policy Enforcement**
- e.g., Prioritize certain RPCs? ➔ **NO**

**Upgradability**
- e.g., Fix vulnerabilities while app running?
Manageability in a feature-rich RPC library?

**Observability**
e.g., How many RPCs? RPC Latency? ➤ YES

**Policy Enforcement**
e.g., Prioritize certain RPCs? ➤ NO

**Upgradability**
e.g., Fix vulnerabilities while app running? ➤ Currently NO
Current Solution to Policy Enforcement

Client
   - Stub
   - RPC Library

Transport
   - NIC
Current Solution to Policy Enforcement

Client
- Stub
- RPC Library

Sidecar
- Policies
- RPC Library

Transport

NIC
Current Solution to Policy Enforcement
Current Solution to Policy Enforcement

Client
- Stub
- RPC Library

Sidecar
- Policies
- RPC Library

Transport

NIC

Server
- Stub
- RPC Library

Transport

NIC

Call

Reply

Marshal

Unmarshal
Current Solution to Policy Enforcement

- **Client**
  - Marshal
  - Unmarshal
  - Stub
  - RPC Library
  - Transport
  - NIC

- **Sidecar**
  - Policies
  - RPC Library
  - Transport
  - NIC

- **Sidecar**
  - Policies
  - RPC Library
  - Transport
  - NIC

- **Server**
  - Stub
  - RPC Library
  - Transport
  - NIC

Call **M**

Reply
Current Solution to Policy Enforcement

**Client**
- Stub
- RPC Library
- Transport
- NIC

**Sidecar**
- Policies
- RPC Library

**Server**
- Stub
- RPC Library
- Transport
- NIC

**RPCLibrary**
- Marshal (M)
- Unmarshal (U)

**Call**
- M
- U

**Reply**
Current Solution to Policy Enforcement

- **Client**: Stub, RPC Library
- **Sidecar**: Policies, RPC Library
- **Server**: Stub, RPC Library

**Components**:
- **Transport**
- **NIC**

**Process**:
- **Call**: marshal, unmarshal, marshal
- **Reply**: unmarshal, marshal, marshal

**Symbols**:
- "M" for Marshal
- "U" for Unmarshal
Current Solution to Policy Enforcement

Client
- Stub
- RPC Library

Transport
- NIC

Sidecar
- Policies
- RPC Library

Server
- Stub
- RPC Library

Call
- Marshal (M)
- Unmarshal (U)

Reply
- Unmarshal (U)
- Marshal (M)
Current Solution to Policy Enforcement

In our evaluation, adding Envoy sidecar to gRPC leads to
- **2.8x** 99th tail latency
- **0.56x** bandwidth (Gbps)

NOT Efficient
RPC-as-a-Library Limitation

- Marshal (M)
- Unmarshal (U)

Client:
- Stub
- RPC Library

Sidecar:
- Policies
- RPC Library

Transport

NIC

Call and Reply flow with Marshal and Unmarshal actions:
- Call: M U M U M U
- Reply: U M U M U M
RPC-as-a-Library Limitation

• RPC library and sidecar are weakly coupled
  • prevent from cross-layer optimization
  • operate/coupled at L4
RPC-as-a-Library Limitation

- RPC library and sidecar are weakly coupled
  - prevent from cross-layer optimization
  - operate/coupled at L4
- RPC Library and app are strongly coupled
  - Difficult to upgrade RPC library
RPC-as-a-Library Limitation

• RPC library and sidecar are weakly coupled
  • prevent from cross-layer optimization
  • operate/coupled at L4

• RPC Library and app are strongly coupled
  • Difficult to upgrade RPC library

We want
• strong coupling: operate at L7
• weak coupling: most of the functionalities extracted into a separate service
mRPC Overview

Client
- Stub
- RPC Library

Sidecar
- Policies
- RPC Library

Transport
NIC

Client to Server
- Call: M U M U M U
- Reply: U M U M U M

RPC-as-a-library

Marshall
- M

Unmarshal
- U
mRPC Overview

Marshall

Unmarshal

Client

Sidecar

Transport

NIC

RPC-as-a-library

RPC-as-a-service: mRPC
mRPC Overview

- **M** Marshal
- **U** Unmarshal

Client
- Stub
- RPC Library

Sidecar
- Policies
- RPC Library

Transport
- NIC

RPC-as-a-library

**RPC-as-a-service: mRPC**

- Client
  - Stub
  - mRPC Library
- Policies
- RPC Library

Transport
- NIC

Client
- Call
- Uni
- M
- M

Server
- M
- Uni

Reply
- U
- M
- M
Challenges

How to support new applications with new RPC specifications at runtime?

How to enforce policies with efficiency and security?

How to live upgrade RPC implementations without disrupting other applications?
Challenges

How to support new applications with new RPC specifications at runtime?

How to enforce policies with efficiency and security?

How to live upgrade RPC implementations without disrupting other applications?
Traditional RPC Libraries

- `Stub`
  - `fn Get()`
  - `fn marshal()`
  - `fn unmarshal()`

- `Proto`

- `App`
  - `Stub`
  - `RPC Library`

- `Service KVStore`
  - `Func Get(GetReq) -> Entry`
In traditional RPC libraries, marshal/unmarshal and service methods code will be generated as a stub and loaded into user applications as a library.
mRPC’s Solution

```
fn Get()
```

1

Compile Stub

```
Service KVStore
Func Get(GetReq) -> Entry
```
mRPC’s Solution

1. Compile Stub

2. Connect(Proto, ...)
mRPC’s Solution

1. Compile Stub
2. Connect(Proto, ...)
3. Compile Marshal Module

mRPC Schema Compiler

App
fn Get()
mRPC Library

mRPC
Marshal
Proto Codegen

Marshal module
fn marshal()
fn unmarshal()

mRPC
Service KVStore
Func Get(GetReq) -> Entry

Proto Marshal module

fn marshal()
fn unmarshal()
In mRPC, marshal/unmarshal code are decoupled from user stub, and generated/loaded by mRPC service instead.
Challenge #2: Memory Management

Service KVStore
Func Get(GetReq) → Entry
Challenge #2: Memory Management

RPC messages are allocated on shared memory heap.

Accessed by both the application and the mRPC service.

Service KVStore
Func Get(GetReq) → Entry

RPC messages are allocated on shared memory heap.

Accessed by both the application and the mRPC service.
Memory Management

Service KVStore
Func Get(GetReq) -> Entry

message_ptr
call_id
func_id

Shared Memory Heap
GetReq

App
Stub
mRPC Library

mRPC
Marshal
Frontend

Desc

Shared Memory Queue
A shared memory queue is used to pass RPC descriptors.
Memory Management

Service KVStore
Func Get(GetReq) -> Entry

message_ptr
call_id
func_id

App
Stub
mRPC Library

mRPC
Marshal
Frontend

Shared Memory Queue
Desc
Desc

Shared Memory Heap
GetReq
Entry

Desc
Memory Management (Outgoing Message)
Memory Management (Outgoing Message)

Diagram:
- **App** (green): Stub, mRPC Library
- **mRPC** (blue): Marshall, Frontend
- **NIC** (black)
- **Shared Memory Heap** (orange): GetReq

1. App (Stub) -> Desc
2. Desc -> mRPC (Marshall)
3. mRPC (Frontend) -> NIC
Memory Management (Outgoing Message)

1. **GetReq**
2. **Completion Notify**
3. **mRPC**
4. **Stub**

**Shared Memory Heap**

**mRPC Library**

**NIC**
Memory Management (Outgoing Message)
Policy Enforcement

Processing Flow / Time

- App
  - Stub
  - mRPC Library
- mRPC
  - Frontend
- mRPC
  - ACL Policy
- NIC

Shared Heap
Policy Enforcement

Processing Flow / Time

App
  Stub
  mRPC Library

mRPC
  Frontend

mRPC
  ACL Policy

key: “dog”

Shared Heap

NIC

Desc
Policy Enforcement

Processing Flow / Time

App
- Stub
- mRPC Library

mRPC
- Frontend

mRPC
- ACL Policy

NIC

Shared Heap

key: “dog”

key: “dog”
Policy Enforcement

Processing Flow / Time

App
- Stub
- mRPC Library

mRPC
- Frontend

mRPC
- ACL Policy

NIC

Shared Heap

key: “dog”

key: “dog”

key: “dog”
Policy Enforcement

Processing Flow / Time

App
- mRPC Library
- Stub

mRPC
- Frontend

mRPC
- ACL Policy

NIC

Shared Heap
- key: “dog”

Private Heap
- key: “dog”
- Copy
Policy Enforcement

Processing Flow / Time

App
- Stub
- mRPC Library
  - Desc: key: “dog”

mRPC
- Frontend
  - Desc: key: “dog”

mRPC
- ACL Policy
  - Desc: key: “cat”
  - Copy
  - App
    - Desc
    - key: “dog”

Shared Heap

Private Heap

NIC

…”dog”
Evaluation

Does mRPC deliver smaller latency and higher goodput compared to existing solutions?

Does mRPC enforce policy efficiently?

Can mRPC improve real-world application’s performance?
Evaluation: Large RPC Goodput

- TCP transport
- Keep 128 concurrent RPCs to hide latency

Speed-up by 3.1x

Evaluated on testbed of servers with 100 Gbps Mellanox Connect-X5 NICs and Xeon 5215 CPUs
## Evaluation: Small RPC Latency

<table>
<thead>
<tr>
<th></th>
<th>Median Latency (μs)</th>
<th>P99 Latency (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>eRPC</td>
<td>3.6</td>
<td>4.1</td>
</tr>
<tr>
<td>mRPC</td>
<td>7.6</td>
<td>8.7</td>
</tr>
<tr>
<td>eRPC + Proxy</td>
<td>11.3</td>
<td>15.6</td>
</tr>
<tr>
<td>mRPC + NullPolicy</td>
<td>7.9</td>
<td>9.1</td>
</tr>
</tbody>
</table>

- RDMA transport
- 64-byte RPC requests, 8-byte replies

Speed-up by 1.7x
Evaluation: Policy Enforcement

- Filter RPCs based on string matching on one field
- 1% requests will not pass

<6% overhead
Evaluation: DeathStarBench

- TCP transport
- Measured over 250 secs @ 20 reqs/sec

Speed-up by 2.5x
Summary

**RPC-as-a-library** cannot meet both **manageability** and **efficiency**

**mRPC:** Reimagined RPC as a **managed system service**

- Efficient policy enforcement
- Upgrade of RPC implementation **without shutting down** user applications

[https://github.com/phoenix-dataplane/phoenix](https://github.com/phoenix-dataplane/phoenix)