RPC File Transfer System

Bui Truong An - 22BA13001

1 Introduction

This report presents the design and implementation of a file transfer system using Remote Procedure Call (RPC) over XML-RPC. The system supports chunk-based file upload, transfer verification, and server-side file listing.

The implementation consists of a Python RPC server and a Python RPC client that communicates using XML-RPC over HTTP.

2 RPC Service Design

2.1 Service Overview

The RPC service exposes the following remote procedures:

- `start_transfer(filename, filesize)`
- `upload_chunk(transfer_id, chunk_data, chunk_number)`
- `finish_transfer(transfer_id)`
- `cancel_transfer(transfer_id)`
- `list_files()`
- `ping()`

Each file transfer session is identified by a unique `transfer_id` which allows the server to track progress.

2.2 RPC Workflow Figure

```
+-----------+             RPC (XML-RPC)             +-------------+
|  Client   | ------------------------------------> |   Server    |
+-----------+                                        +-------------+
      |                                                    |
      | start_transfer(filename, filesize)                 |
      |--------------------------------------------------->|
      |                                                    |
      | upload_chunk(transfer_id, base64data, n)           |
      |--------------------------------------------------->|
      |                                                    |
      | finish_transfer(transfer_id)                       |
      |--------------------------------------------------->|
      |                                                    |
      | <----------------------- success / fail -----------|
```

This workflow ensures reliable transfer using chunk-by-chunk upload with base64 encoding.

3 System Organization

3.1 Modules

The system is divided into two core components:

- **RPC Server**: Handles storage, chunk writes, and file assembly.
- **RPC Client**: Reads files, encodes chunks, and uploads them.

3.2 System Architecture Figure

```
+--------------------------------------------------------------+
|                          RPC Server                          |
|--------------------------------------------------------------|
|  - start_transfer()                                          |
|  - upload_chunk()                                            |
|  - finish_transfer()                                         |
|  - cancel_transfer()                                         |
|  - list_files()                                              |
|                                                              |
|  File Storage: /uploads                                      |
+--------------------------------------------------------------+


                | XML-RPC over HTTP


+--------------------------------------------------------------+
|                          RPC Client                          |
|--------------------------------------------------------------|
|  - connect()                                                 |
|  - send_file()                                               |
|  - list_server_files()                                       |
+--------------------------------------------------------------+
```

4 Implementation

This section shows the essential parts of the code implementing the file transfer.

4.1 Server-Side Code Snippets

4.1.1 Start Transfer

```
def start_transfer(self, filename, filesize):
    transfer_id = f"{filename}_{datetime.now().strftime('%Y%m%d_%H%M%S')}"
    save_filename = f"received_{timestamp}_{filename}"

    filepath = os.path.join(self.upload_dir, save_filename)
    self.active_transfers[transfer_id] = {
        'filename': filename,
        'filepath': filepath,
        'filesize': filesize,
        'received': 0,
        'file_handle': open(filepath, 'wb')
    }

    return {'status': 'success', 'transfer_id': transfer_id}
```

4.1.2 Upload Chunk

```
def upload_chunk(self, transfer_id, chunk_data,
   chunk_number):
    transfer = self.active_transfers[transfer_id]
    binary_data = base64.b64decode(chunk_data)

    transfer[’file_handle’].write(binary_data)
    transfer[’received’] += len(binary_data)

    return {’status’: ’success’, ’progress’: progress}
```

4.1.3 Finish Transfer

```
def finish_transfer(self, transfer_id):
    transfer = self.active_transfers[transfer_id]
    transfer[’file_handle’].close()

    actual_size = os.path.getsize(transfer[’filepath’])

    if actual_size == transfer[’filesize’]:
        del self.active_transfers[transfer_id]
        return {’status’: ’success’, ’filepath’: transfer[’
           filepath’]}
```

4.2 Client-Side Code Snippets

4.2.1 Connect to Server

```
self.proxy = xmlrpc.client.ServerProxy(self.server_url,
   allow_none=True)
response = self.proxy.ping()
```

4.2.2 Start File Transfer

```
response = self.proxy.start_transfer(filename, filesize)
transfer_id = response[’transfer_id’]
```

4.2.3 Upload File Chunks

```
with open(filepath, ’rb’) as f:
    while True:
        chunk = f.read(self.chunk_size)
        if not chunk:
            break

        encoded_chunk = base64.b64encode(chunk).decode(’utf-8’)
        self.proxy.upload_chunk(transfer_id, encoded_chunk, chunk_number)
```

4.2.4 Finish Transfer

```
response = self.proxy.finish_transfer(transfer_id)
if response[’status’] == ’success’:
    print("Transfer␣completed!")
```

5 Conclusion

The RPC-based file transfer system successfully supports large file uploads by splitting them into base64-encoded chunks. This design ensures that transfers can be tracked, canceled, and verified on the server. The architecture cleanly separates responsibilities between the RPC server and the client.