Project Description

Description

Currently, rFaaS uses the ibverbs API for RDMA operations to achieve high performance and low-latency remote memory access. However, the ibverbs API is limited in the types of network protocols it supports. To support other types of network protocols, we need to use a higher-level communication library called libfabric for RDMA operations while maintaining compatibility with the existing ibverbs implementation. This project has two main goals:

1. Introduce the libfabric library and refactor the existing codebase to use a compatible implementation of ibverbs and libfabric.
2. Adjust the libfabric configuration for TCP or AWS EFA protocols to support a wider range of network types.

The Scope of Work

- Familiarity with C/C++.
- Familiarity with RDMA operations, such as registering and deregistering memory regions, creating and destroying QPs, initiating and receiving data transfer requests, error handling, etc.
- Understanding of the working principles and application scenarios of different network protocols, such as InfiniBand, RoCE, TCP/IP, and AWS EFA.
- Understanding of how to use the libfabric library for RDMA operations.
- Understanding of how to configure and use the ibverbs, sockets provider, and EFA provider libraries.
- Familiarity with writing and using test cases, optimizing them to ensure code reliability and performance.

Complexity

- The libfabric API is complex and has a high learning curve.
- Different network protocols have their own characteristics, resulting in a broad scope of work.
- When using different network protocols, the specific libfabric operations required are not completely the same, and it is necessary to handle these differences properly.
- Performance optimization requires selecting the optimal solution based on many different application scenarios to ensure performance.
- Guaranteeing reliability requires careful consideration of error handling when modifying the code to support multiple different network protocols and RDMA adapters.

Related Work

Open MPI uses the communication interface provided by libfabric to allow applications to switch between different transport layers. When using network protocols supported by ibvers, libfabric uses ibverbs as the underlying transport protocol for data transfer, thus providing high performance and low-latency communication. When using network protocols not supported by ibvers, other libfabric providers are used for RDMA operations.

Project Proposal

Libfabric is an advanced networking library that abstracts the underlying network via adapter endpoints and provides a uniform programming interface for different network protocols to enable high-performance and low-latency RDMA operations. Similar to Open MPI, we can make use of the libfabric API directly in programs and establish a bridge between the program and hardware through the libfabric provider, enabling communication via the libfabric API with different hardware and software platforms.

We need to refactor the existing code that uses ibverbs interface for network communication, replacing it with the libfabric library and utilizing the unified interface provided by libfabric to achieve the previous functionality. Additionally, we need to introduce new libfabric APIs to implement features and functionality that ibverbs does not have.

For different network environments, we need to write code that allows the client and server to perform different initialization configurations and establish connections with each other based on the current network protocol in use. Libfabric selects an appropriate provider, such as verbs, sockets, PAMI, etc., as the underlying network transport layer to complete subsequent remote memory access operations.

The end result is that the client and server select the optimal underlying network transport layer as the provider based on the currently supported network protocol and undergo appropriate initialization configuration before establishing a connection. The client and server then utilize the libfabric interface to complete RDMA operations via the underlying network transport layer and release resources before exiting.

We can still utilize the existing code framework to accomplish the above operations by simply replacing the previously used ibverbs interface with the libfabric interface.

The following are the main code refactoring tasks:

Existing Code	Original Function	Major Changes	Possible API Used
Address class	Represents the IP address and port of the RDMA connection.	Modify the variable used and initialize the necessary parameters. Use the APIs in fi_av_set and fi_av to manage IP addresses and ports.	`fi_av_open`, `fi_close` `fi_av_bind`, `fi_av_insert` `fi_av_set`, `fi_close`...

Existing Code	Original Function	Major Changes	Possible API Used
RDMAActive class and RDMAPassive class	Actively connect to another node. Passively receive connection requests from other nodes.	Merge into a single Connection class (see table below).	N/A
Connection class	Conducts RDMA communication, including initialization, message publishing/receiving, event polling, closing connections, batch receiving, publishing writes, and atomic operations.	Renamed to Operations class. Remove initialization and connection closing operations. Modify the variable used and initialize the necessary parameters. Use the APIs in fi_tagged and fi_msg to publish buffers for incoming messages and initiate sending messages. Use the APIs in fi_rma to complete remote memory read and write operations. Use the APIs in fi_poll to complete event polling. Use the APIs in fi_atomic to complete atomic operations on remote memory.	`fi_trecv`, `fi_tsend` `fi_recv`, `fi_recvv` `fi_send`, `fi_sendv` `fi_read` `fi_write`, `fi_readv` `fi_writev`, `fi_poll_open` `fi_poll`, `fi_wait` `fi_atomicv` `fi_fetch_atomic`...
Buffer class	Manages the registration and deregistration of local and remote memory.	Modify the variable used and initialize the necessary parameters. Use the APIs in fi_mr to manage local and remote memory.	`fi_mr_reg`, `fi_close` `fi_mr_key`, `fi_mr_raw_attr` `fi_mr_refresh`...

Existing Code	Original Function	Major Changes	Possible API Used
recv_buffer structure	Represents the receive buffer used with ibverbs.	Modify the variable used and add parameters. Change to a buffer structure that is compatible with libfabric and use the corresponding buffer type and operations based on the currently used libfabric provider. For example: unnamed socket buffer, Shared Receive Queue buffer, etc.	`socket()`, `fi_recvmsg()` `fi_sendmsg()`, `fi_srq_open()` `fi_srq_set_attr()`...
executor remote function call executor	Loads shared libraries, allocates and releases resources, receives and sends data.	Modify the variable used. Use the APIs in fi_eq and fi_cntr to load shared libraries. Access libfabric resources through the Fabric class operations. Use the Connect class operations to implement data read and write operations.	`fi_eq_open`, `fi_close` `fi_eq_write`, `fi_eq_read` `fi_cntr_open`, `fi_cntr_add`, `fi_writedata` `fi_readv`...
fast_executor function computing service program	Continuously polls the queue, executes function calculations, and returns results to the client.	Modify the variable used. Use the APIs in fi_cq, fi_eq, fi_cntr, and fi_poll to poll the events queue. Use the Connect class operations to implement data read and write operations.	N/A

Aside from the above code, some other code also needs to be modified, such as the server and client code. In this code, we simply use libfabric APIs and variables instead of the ones previously used. Note that different providers have different resource control capabilities, and hence we use different libfabric APIs depending on the provider used.

Here is the optimized markdown proposal for submission to Google's GSoC program:

Possible Code Additions

New Code	Functionality	Main Implementation Approach	Possible API Used
`Fabric` Class	Stores information such as the supported network protocols, available providers, selected adaptors, Fabric object, domain object, and endpoint object. Retrieves the `fi_info` structure and the saved information inside it. Creates, closes, and manages Fabric, domain, endpoint objects, as well as other related operations.	Uses APIs in `fi_getinfo` to retrieve interface information. Uses APIs in `fi_fabric` to create and close Fabric objects, etc. Uses APIs in `fi_domain` to create and close domain objects. Uses APIs in `fi_endpoint` to create and close endpoints.	`fi_getinfo`, `fi_freeinfo` `fi_fabric`, `fi_close` `fi_domain`, `fi_domain_bind` `fi_endpoint`, `fi_ep_bind` `fi_enable`, etc.
`Connection` Class	Stores information required for establishing a connection. Manages the endpoint connection state. Allows an endpoint to actively connect to another node. Allows an endpoint to listen for connections from other endpoints and passively receive connections from other endpoints. Returns information about the endpoint and connection.	Stores required struct and variable information and initializes parameters. Uses APIs in `fi_cm` to establish connections and return endpoint and connection information.	`fi_connect`, `fi_listen` `fi_accept`, `fi_shutdown` `fi_getname`, etc.
`errno` Error	Converts Fabric errors to printable strings.	Uses APIs in `fi_errno`.	`fi_strerror`

Timeline

Week	Dates	Work
0	Before May 29	Get familiar with the project, continue contributing to the project via pull requests, read Libfabric man pages to gain mastery over the usage of Libfabric API, refine the project framework and code implementation, and understand the principles and implementation of different network protocols and required Libfabric providers. Prepare for writing code.

Week	Dates	Work
1-2	May 29th - June 11th	Introduce the Libfabric library and complete preliminary coding of Fabric and Address classes.
3-4	June 12th - June 25th	Complete preliminary coding of Buffer and Connection classes.
5-6	June 26th - July 9th	Complete initial coding of `recv_buffer` struct, executor remote function call executer, and `fast_executor` function computing service program.
7-8	July 10th - July 23th	Complete preliminary coding and modifications of server, client, and other related code.
9-10	July 24th - August 6th	Perform preliminary improvement of the overall code to ensure normal program operation when using `verbs` as provider. Conduct testing and code optimization to ensure program reliability and performance.
11-12	August 7th - August 20th	Finish the overall code to ensure that the program runs properly when using `sockets`, `efa` (for TCP and AWS EFI), and other providers. conduct testing and code optimization to ensure program reliability and performance.

Buffer Time (8 days)

Allow for a buffer period of seven days as a safety net, in case anything does not go as planned.

About me

Personal Information

- Name: Ye Yuan
- Major: Network Engineering
- University: Xi'an University of Posts
- Degree: Undergraduate sophomore
- Email: origin020726@gmail.com
- Github: https://github.com/Origin-yy
- TimeZone: Shanghai, China (GMT+8)

Schedule

Before the summer vacation of school (around July 15th), I can dedicate 4-6 hours of free study time every day, averaging more than 30 hours per week, in addition to school courses. Of course, there will be mid-term and final exams.

After the summer vacation, according to the group's habits, I will stay on campus to study and spend the whole summer studying in the group. We have a scheduled timetable, with 8+ hours of group study every day from Monday to Saturday, and the study content is arranged by ourselves. There are no requirements on Sundays, and there will be one outing activity.

During the entire GSoC project period, I have no other major activities planned, and most of this free time will be used to learn technology, with completing the GSoC project as the top priority.

Educational Experience

In September 2021, I entered the Network Engineering major at Xi'an University of Posts and Telecommunications and began to study computer network-related knowledge. In November 2021, I became a member of XiYou Linux Group and started to learn about Linux. From 2022 until now, I have studied Linux system programming and network programming, including Linux system calls, multi-threading and multi-processing, thread pools, TCP/IP protocol, I/O multiplexing, socket communication, and transitioned from C to C++.

Code Experience

- Familiarity with common data structures and algorithm problems.code link
- Independently implemented the "ls" command using C language and Linux system calls, supporting flexible combinations of parameters such as -a, -l, -s, -t, -r, -i, and -R.code link
- Independently implemented a shell using C language, inter-process communication and Linux system calls, supporting operations such as pipes, redirects, background running, and changing directory.code link
- implemented producer-consumer and dining philosophers problems using C multi-threading, as well as a thread pool.code link
- Independently developed a C++ multi-user network chat room on a LAN using socket, C++thread pool, I/O multiplexing, and redis. The clients and servers are highly stable and concurrent, supporting common operations such as group chat, adding/deleting friends, file sending, and offline messaging, with data persistence using redis.code link

Others

Why did you choose this project? What made you interested in our organization?

rFaaS supports the high-performance, low-latency calling of network resources using RDMA. The relevant technology required for this project aligns with my learning direction, and I have a strong interest in it.

What do you wish to accomplish during GSoC?

I hope to learn new technologies that I'm interested in, improve my technical and communication collaboration abilities, and make contributions to open-source projects that I am passionate about.

Have you worked with the required technologies before? If yes, then in what scope? Which projects?

I have not used ibverbs, libfabric, or RDMA before. However, I have some knowledge of network programming, TCP/IP, C++, and thread pools through learning, which I gained in XiYou Linux Group, and can be found on my GitHub.

Did you contribute to open-source projects before? Have you ever made a pull request, helped to fix an issue, or developed a project?

No, this is my first time participating in an open source project.