Enabling Remote Direct Memory Access in XRootD

Patrick Adolf, Nazar Burmasov

XRootD overview

Remote Direct Memory Access mechanism

The librdmacm library

OpenFabrics Interfaces

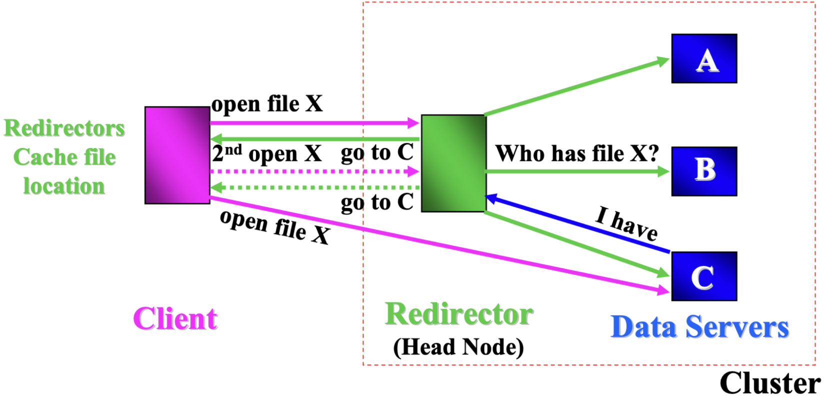
What is XRootD?

- Also called «data access system»
- Fully generic suite for fast, low latency and scalable data access
- High performance, scalable, fault tolerant access to data repositories of many kinds
- Can serve nearly any kind of data:
  - Organized as a hierarchical filesystem-like namespace
  - Based on the concept of directory
- Plugin-based and open
- Dynamic file discovery
  - No record where the files are

What can be done with XRootD?

- One network connection per client for many open files
- Scalable to hundreds of servers
- Allows to reuse files descriptors between clients
- Widely used in the High Energy Physics community
- HEP experiments start using the HPC infrastructure
  - More important for the framework to support Remote Direct Memory Access

Overview



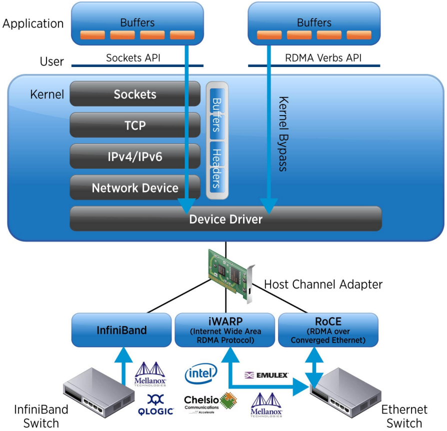
Redirectors
Cache file
location
open file X
2nd open X
go to C
go to C
open file X
Who has file X?
I have
A
B
C
Client
Redirector
(Head Node)
Data Servers
Cluster

Slurm

- CERN Linux Cluster
- Multi node execution and fast interconnects
- Over 65 000 physical nodes
- Small testing cluster for our project
  - IWARP test setup with two nodes
- Submit jobs to the HPC queue with 'srun' utility
- Process being stack in D-state
  - One of the servers gets inaccessible

Remote Direct Memory Access

- **«Traditional» data transfer**: data must go through buffers using the sockets API → involves TCP, IP stack, device driver → CPU overhead
- **RDMA**
  - OS interventions and buffers are avoided
  - Zero-copy networking – no work done by CPUs
  - High throughput, low-latency communications → good for high-performance computing
  - Needs dedicated Host Channel Adapters



librdmacm

- Library for managing RDMA connections
  - Allows applications to set up reliable connected and unreliable datagram transfers over RDMA
- Transport-neutral interface
  - Same code can be used for both InfiniBand and IWARP adapters
  - Huge benefit
- Interface based on sockets; adapted for queue pair based semantics
- Communication must use a specific RDMA device
- Data transfers are message-based

librdmacm

- librdmacm only provides communication management
  - Connection setup and tear-down
- librdmacm is build on top of libverbs
  - Responsible for carrying out the transfer

How to send data with librdmacm

- Focus on a minimalistic example code to transfer a buffer
  - Base for further developments
- Link: code on Github
- Uses build.sh script to build the project
- Usage:
  - Server: `srun -w [hostname] -p batch-short -t 00:05:00 -n 1 /path/to/our/code/rcopy`
  - Client: `srun -w [hostname] -p batch-short -t 00:05:00 -n 1 /path/to/our/code/rcopy [servername]`
- Link: rcopy.c code

rcopy.c

- Main function is splitted into a server and a client code
- server_opt and client_opt are only for parsing command line arguments

```c
int main(int argc, char **argv){
	int ret;
	if (argc == 1 || argv[1][0] == '-') {
		server_opts(argc, argv);
		ret = server_run();
	} else {
		client_opts(argc, argv);
		ret = client_run();
	}
	return ret;
}
```

Server Function

```
static int server_run(void)
{
        int lrs, rs;
        union rsocket_address rsa;
        socklen_t len;

        lrs = server_listen();    // checking server address
        if (lrs < 0)
                return lrs;
        len = sizeof rsa;
        printf("waiting for connection...");
        fflush(NULL);
        rs = raccept(lrs, &rsa.sa, &len);   // connecting server
        printf("client: %s\n", _ntop(&rsa));

        char buffer[17];        // buffer for the recieving data from client
        memset( buffer, 0, sizeof(buffer) );
        _recv(rs, buffer, sizeof(buffer)-1 ); //Recieving the sending data from client
        buffer[16] = 0;
        printf("received buffer: %s", buffer);

        char buffer2[16];  // second buffer for own data sending to client
        memset( buffer2, 'B', sizeof( buffer2 ) );
        len = rsend(rs, buffer2, sizeof( buffer2 ), 0); //Sending data with information about size and socket tu client
        printf( "send %d bytes", len );

        rshutdown(rs, SHUT_RDWR);
        rclose(rs); //end connection
        return 0;
}
```

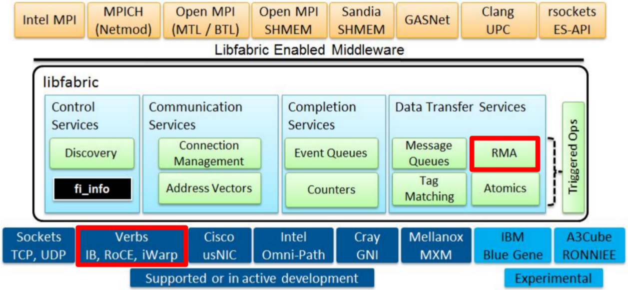
Client Function

```
static int client_run(void)
{
        struct msg_hdr ack;  // declaring some needed variables
        int ret, rs;
        size_t len;

        rs = client_connect();  // connecting client
        if (rs < 0)
                return rs;
        printf("...");
        fflush(NULL);
        gettimeofday(&start, NULL);

        char buffer[16];    // buffer with own data to send to server
        memset( buffer, 'A', sizeof( buffer ) );
        len = rsend(rs, buffer, sizeof( buffer ), 0); //Sending own data with information about socket and size
        printf( "send %d bytes", len );

        char buffer2[17];      // empty buffer for the data recieving from server
        memset( buffer2, 0, sizeof(buffer2) );
        _recv(rs, buffer2, sizeof(buffer2)-1 ); //Getting data from server
        buffer2[16] = 0;
        printf("received buffer: %s", buffer2);
        gettimeofday(&end, NULL);

shutdown:
        rshutdown(rs, SHUT_RDWR);
        rclose(rs); // end connection
        return 0;
}
```

OpenFabrics Interfaces

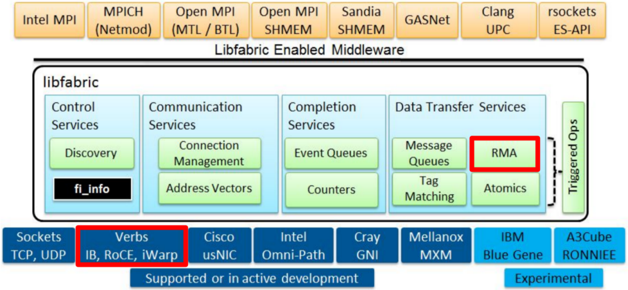
The libfabric library

- Provides interfaces to a number of hardware solutions and some basic services
- In most cases, applications access the provider implementation directly → low latency
- Providers: hardware specific; plug into the framework to provide access to the fabric hardware



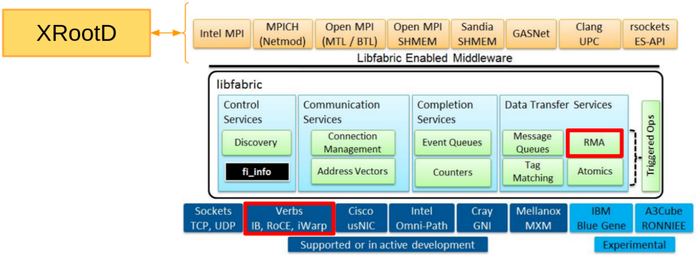
Main parts of the libfabric:

- **Control services**: discover system communications
- **Communication services**: setup communication between nodes
- **Completion services**: report results of operations (event queues or counters)
- **Data transfer services**: set of interfaces for different communication approaches



Enabling XRootD:

- Need to implement a simple example for further developments



RDMA file copying with libfabric

- Simple file transfer using libfabric and some modified utilities from fabtests
  - Github repository: libfabric
  - Github repository: fabtests
- Built with CMake to enable automatic fetching of in-system libfabric version
- Simple usage:

  Client: `./oficp -r[ead] /full/path/to/file -y[ank] /full/path/to/copy hostname`

  Server: `./oficp`
- The full code is available on Github

Client code

- `ft_get_tx_comp()` is implemented in terms of `fi_cntr_read()` and `fi_cntr_wait()`, which allows interacting with the completion queue

```
FILE* file_ptr = fopen(opts.src_filename, "rb");          // open the file for reading
fseek(file_ptr, 0, SEEK_END);                              // get file's size
size_t file_len = ftell(file_ptr);
snprintf(tx_buf, tx_size, "%s", opts.dst_filename);
int name_len = strlen(opts.dst_filename) + 1;
(void) fi_write(ep, tx_buf, name_len, mr_desc,             // send destination
                remote_fi_addr, remote.addr, remote.key,   // file name to the server
                &fi_ctx_write);
(void) ft_get_tx_comp(++tx_seq);
char file_len_ch[256];
snprintf(file_len_ch, sizeof(file_len_ch), "%zu", file_len);
snprintf(tx_buf, tx_size, "%s", file_len_ch);
fi_write(ep, tx_buf, sizeof(file_len_ch), mr_desc,        // send file size to the server
         remote_fi_addr, remote.addr, remote.key,
         &fi_ctx_write);
(void) ft_get_tx_comp(++tx_seq);
rewind(file_ptr);
tx_buf = (char*)malloc((chunk_size + 1) * sizeof(char)); // allocating chunk
size_t n_transx = file_len / chunk_size;                  // number of transactions
if (file_len % chunk_size != 0)
  n_transx++;
for (size_t itx = 0; itx < n_transx; itx++) {             // read and send the file chunk by chunk
  for (size_t i = 0; i < chunk_size; i++)
    tx_buf[i] = '\0';
  fread(tx_buf, chunk_size, 1, file_ptr);
  (void) fi_write(ep, tx_buf, chunk_size, mr_desc,
                  remote_fi_addr, remote.addr, remote.key,
                  &fi_ctx_write);
  (void) ft_get_tx_comp(tx_seq++);
}
fclose(file_ptr);                                         // close the file and wait
(void) ft_get_rx_comp(rx_seq++);                          // for the success message
fprintf(stdout, "Received data from Server: %s\n", (char*)rx_buf);
```

Server code

- `ft_get_rx_comp()` is also implemented in terms of `fi_cntr_read()` and `fi_cntr_wait()`

```c
(void) ft_get_rx_comp(rx_seq++);                          // receive destination file name and size
opts.dst_filename = (char*)rx_buf;                        // and prepare to write
FILE* dst_file_ptr = fopen(opts.dst_filename, "wb");
size_t file_len;
for (size_t i = 0; i < rx_size; i++)
  rx_buf[i] = '\0';
(void) ft_get_rx_comp(rx_seq++);
file_len = atol(rx_buf);
size_t n_transx = file_len / chunk_size;
if (file_len % chunk_size != 0)
  n_transx++;
for (size_t irx = 0; irx < n_transx; irx++) {             // receive and write the file
  for (size_t i = 0; i < chunk_size; i++)                 // chunk by chunk
    rx_buf[i] = '\0';
  (void) ft_get_rx_comp(rx_seq++);
  fwrite((char*)rx_buf, chunk_size, 1, dst_file_ptr);
}
fclose(dst_file_ptr);                                    // close the file and send
char* fin_message = "Success: end of transmission"; // the completion message
size_t msg_size = strlen(fin_message);
snprintf(tx_buf, tx_size, "%s", fin_message)
(void) fi_write(ep, tx_buf, msg_size, mr_desc,
                remote_fi_addr, remote.addr, remote.key,
                &fi_ctx_write);
(void) ft_get_tx_comp(tx_seq++);
```

Testing file copying

- Test data: English Wikipedia dump (2006)

  enwik8 – $100 \cdot 10^6$ bytes
  enwik9 – $1000 \cdot 10^6$ bytes
- The chunk size per transfer has to be defined manually – using 10^6 bytes for this test

Client

```
nburmaso hpc003 /hpcscratch/user/nburmaso/project/oficp/build
→ ./oficp -r /hpcscratch/user/nburmaso/project/oficp/data/text_samples/enwik8 -y /hpcscratch/user/nburmaso/project/oficp/data/enwik8_copy hpc002
Trying to read a file: /hpcscratch/user/nburmaso/project/oficp/data/text_samples/enwik8
file_len = 100000000
RMA write to server, chunk #0
RMA write to server, chunk #1
RMA write to server, chunk #2
(...)
RMA write to server, chunk #99
Received data from Server: Success: end of transmission
```

Server

```
nburmaso hpc002 /hpcscratch/user/nburmaso/project/oficp/build
→ ./oficp
Received data from Client: /hpcscratch/user/nburmaso/project/oficp/data/enwik8_copy
Opening file for writing: /hpcscratch/user/nburmaso/project/oficp/data/enwik8_copy
file_len = 100000000
Received data from Client: #0
Received data from Client: #1
Received data from Client: #2
(...)
Received data from Client: #99
```

Conclusion

- RDMA is a powerful mechanism, which can be used to implement a high-throughput, low-latency infrastructure
- Adopting librdmacm for XRootD is a low-handing fruit
- A simple file transfer is implemented using libfabric, and it can be used for further developments