Regular paper | Journal of the KIECS. pp. 1133-1144, vol. 16, no. 6, Dec. 31. 2021, t. 110, pISSN 1975-8170 | eISSN 2288-2189
http://dx.doi.org/10.13067/JKIECS.2021.16.6.1133

대용량 데이터의 전송 효율 및 기록 성능 향상을 위한 Zero Copy 기술 적용에 관한 연구

송민규* · 김효령* · 강용우* · 제도흥* · 위석오* · 이성모* · 김승래*

A Study on the Application of Zero Copy Technology to Improve the Transmission Efficiency and Recording Performance of Massive Data

Min-Gyu Song* · Hyo-Ryoung Kim* · Yong-Woo Kang* · Do-Heung Je* · Seog-Oh Wi* · Sung-Mo Lee* · Seung-Rae Kim*

요 약

Zero-copy는 메모리 무복사로도 불리는 기술로서 이에 대한 사용을 통해 사용자 영역과 커널 영역 간 컨텍스트 스위칭을 줄여 CPU의 부하를 최소화할 수 있다. 하지만 이 기술은 소규모의 랜덤한 파일을 전송하는 용도에 그치고, 대용량 파일 전송에는 아직 널리 활용되지 못하고 있다. 본 논문은 네트워크를 경유한 대용량 파일 처리에 있어서 Zero-copy의 실질적인 적용 방안에 대해 논의하고자 한다. 이를 위해 먼저 Zero-copy 기반으로 데이터를 전송, 저장할 수 있는 소규모 테스트베드 구축 및 프로그램을 개발하였다. 이후 세부 성능 평가를 통해 적용된 기술의 유용성을 세부 검증하고자 한다.

ABSTRACT

Zero-copy is a technology that is also called no-memory copy, and through its use, context switching between the user space and the kernel space can be reduced to minimize the load on the CPU. However, this technology is only used to transmit small random files, and has not yet been widely used for large file transfers. This paper intends to discuss the practical application of zero-copy in processing large files via a network. To this end, we first developed a small test bed and program that can transmit and store data based on zero-copy. Afterwards, we intend to verify the usefulness of the applied technology in detail through detailed performance evaluation

키워드

Zero-Copy, TCP/IP Socket, High Speed Network, Data Transfer, Storage
제로 카피, TCP/IP 소켓, 초고속 네트워크, 데이터 전송, 스토리지

Ⅰ. 서 론

IT 기술의 급속한 발전 속에 온라인을 이용한 데이터 전송의 비중이 갈수록 커지고 있다. IT 컨설팅 회사 IDC에 따르면 10년 전과 비교해 네트워크 성능은 10배 이상 증가했고 인터넷 상에서 데이터 양은

* 한국천문연구원 (hrkim, byulmaru, dhje, sowi, vsat, nvibri@kasi.re.kr)
* 교신저자 : 한국천문연구원 전파천문본부
· 접 수 일 : 2021. 10. 18
· 수정완료일 : 2021. 11. 17
· 게재확정일 : 2021. 12. 17
· Received : Oct. 18, 2021, Revised : Nov. 17, 2021, Accepted : Dec. 17, 2021
· Corresponding Author : Min-Gyu Song
Division of Radio Astronomy, Korea Astronomy and Space Science Institute
Email : mksong@kasi.re..kr

50배 이상 늘어난 것으로 분석된다[1]. 이러한 패턴은 VLBI 분야 역시 크게 다르지 않다. 국내의 경우, 불과 10년 전까지만 하더라도 KVN 각 사이트에서 획득한 관측 데이터는 디스크 팩에 저장되어 택배 등의 방식으로 운송되었다. 하지만 2010년 8월 1GbE 회선 개통을 시작으로 10GbE, 40GbE로 네트워크 성능이 단계적으로 개선되어 내년에는 세 사이트와 상관센터 각각 40GbE, 200GbE 성능의 업그레이드를 눈앞에 두고 있다[2]. 이러한 현실 속에 KVN 각 사이트의 방대한 데이터 파일을 상관센터의 스토리지로 신속하게 전달하는 것이 중요한 요소로 부각되고 있다. 이를 구현함에 있어서 적용할 수 있는 여러 기술이 있는데 TCP 전송의 경우 병렬 스트림이 널리 사용되고 있다[3]. 하지만 이는 네트워크 성능이 불안정할 경우 유용한 방식으로 정상적인 네트워크라면 단일 스트림과 병렬 스트림의 데이터 전송 성능 간의 차이는 없어야 한다[4]. 또다른 방식으로는 컴퓨팅 메커니즘의 효율성 제고를 통한 CPU 부하 경감이 있는데 CPU 영역 - 사용자 영역 간 불필요한 컨텍스트 스위칭 감소를 통해 전송 성능을 향상시키는 것을 목적으로 한다[5]. 이 중 본 논문에서는 후자에 초점을 맞춰 성능 개선 방안 대해 논의하기로 한다.

Zero-copy는 랜덤한 형태의 소규모 데이터 파일에서 그 유용성이 충분히 입증되었다. 한 예로, 128KB을 단위 크기로 하는 파일의 경우 Zero-copy를 사용하면 전송 성능을 50% 가량 향상시킬 수 있고 효율적인 CPU 관리가 가능하다[6]. 하지만 대용량 파일에서는 아직 폭넓게 적용되지 않고 있으며, 정량화된 성능 역시 구체적으로 보고된 것이 없다. 이에 따라 본 논문에서는 해당 기술의 실제 적용을 통해 대용량 데이터 파일의 전송과 저장을 구현해 보고자 한다. 이를 위해서는 프로그램 제작, 테스트베드 구축이 전제되어야 한다. 또한 실험을 통해 Zero-copy가 대용량 데이터 파일의 전송, 저장을 위한 유용한 수단인지 검증해 보고자 한다.

이를 위한 논문의 구성은 다음과 같다. 서론에 이어 2장에서는 본 논문의 기반 기술에 해당하는 zero-copy에 대해 살펴볼 것이다. 3장에서는 성능 향상을 목표로 시스템 설계, 개발에 대해 자세히 알아보고 4장에서 성능 평가를 위한 테스트베드 구축과 각종 실험 진행 그리고 결과에 대해 세부적으로 기술할 것이다. 그리고 5장에서 본 논문의 결론을 맺고자 한다.

II. Zero-copy 개요 및 연구 범위

Zero-copy는 데이터 처리 과정에서 CPU로 전달되는 부하를 줄여 성능 향상을 구현할 목적으로 제시된 기술이다. 하지만 애초의 취지와 달리 Zero-copy는 대용량 데이터 등 범용적으로 활용되지 못하고 있는데 본 장에서는 Zero-copy의 개요와 기술적 한계에 대해 살펴보고자 한다.

2.1 Zero-copy 개요

컴퓨터 네트워크 상에서 데이터 전송을 위해 널리 사용되는 라이브러리는 Socket API이며 임의의 클라이언트에서 파일을 전송할 경우 read(int fd, void *buf, size_t count) 함수를 통해 파일로부터 메모리로 데이터를 저장한다. 메모리 상의 해당 데이터는 이후 write(int fd, void *buf, size_t count) 함수 호출을 통해 소켓에 지정된 네트워크 상의 목적지 서버로 전송된다. 하지만 이를 통한 데이터 전송 과정에서 커널 영역 - 사용자 영역 간 다수의 컨텍스트 스위칭과 메모리 복사가 발생하는데, 컨텍스트 스위칭은 수행

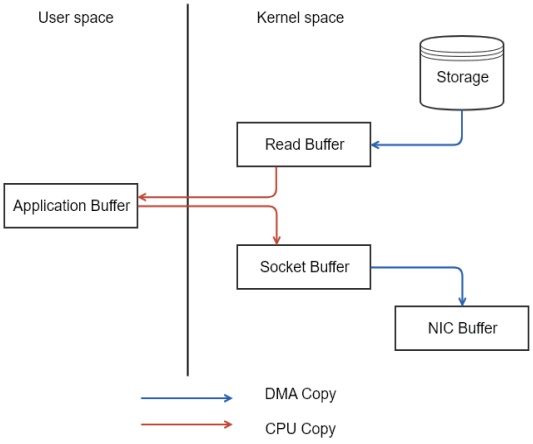

그림 1. 저수준 함수 호출을 통한 기존의 파일 전송 및 그 과정에서의 컨텍스트 스위칭
Fig. 1 Conventional file transfer through low-level function calls and context switching in the process

정보들을 백업하는 등의 오버헤드를 유발하고, 빈번한 메모리 복사는 CPU 자원의 소모로 이어진다[7].

기존의 파일 전송의 경우 그림 1에서 알 수 있듯이 각각 4번의 컨텍스트 스위칭과 메모리 복사가 발생한다. 따라서 성능 향상을 위해서는 이를 최소화하는 것이 필요하다.

아래 그림 2는 이러한 문제점을 극복하기 위해 제안된 Zero-copy의 작동 기제에 관한 것으로 DMA 엔진이 디스크에서 파일을 읽어 커널에 위치한 Read buffer로 데이터를 복사하는 점은 동일하다. 하지만 해당 데이터는 사용자 영역의 애플리케이션 버퍼를 거치지 않고 Read buffer 상의 데이터 위치와 사이즈 정보를 이용해 NIC 버퍼로 바로 복사되고, 이후 네트워크를 경유해 소켓에 지정된 목적지 주소로 전송된다. 이를 통해 컨텍스트 스위칭과 메모리 복사는 2번씩으로 줄어들어 성능 향상 효과를 얻을 수 있다.

2.2 연구 범위

Zero-copy 기법은 실질적인 대용량 파일 전달에 활발히 적용되지 못하고 있고 소용량 데이터에 대한 반복적인 호출을 통해 전송 성능을 측정하는 용도로 널리 쓰이고 있다. IBM의 실험 사례를 보더라도 1GB 이하의 데이터 용량에 그치고 있다는 점에서 활용이 제한적이다[8]. 데이터 전송 이후 실제 디스크 상의 저장 성능을 측정한 자료 역시 찾기 어려운 실정인데,

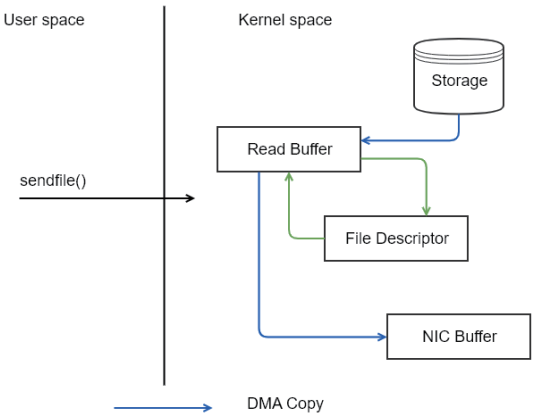

그림 2. sendfile 함수 호출을 통한 zero-copy 구현 및 그 과정에서의 컨텍스트 스위칭
Fig. 2 Zero-copy implementation through sendfile function call and context switching in the process

이는 사용자로 하여금 zero-copy의 실질적 활용을 제한하는 요소로 작용한다. 이에 따라 본 논문에서는 크게 두 가지 목표 하에 연구를 진행하였다. 먼저 전달 대상에 해당하는 데이터의 용량을 100GB로 증가시켰고 이를 zero-copy 기반으로 네트워크로 전송하는 것을 1차 목표로 삼았다. 또한 단순 스트림 전송에 그치지 않고 최종단에 위치하는 스토리지에 파일로 저장하는 것을 2차 목표로 설정하였고 이에 대한 성능 비교 분석을 통해 zero-copy의 유효성을 실질적으로 검증하였다.

III. 파일 전송 프로그램 개발

본 장에서는 파일 전송 성능을 향상시키기 위한 시스템 설계, 개발에 대해 알아보기로 한다. 이를 위해 TCP/IP 소켓 API에 기반한 프로그램을 먼저 설계하고 데이터 송수신 관련 주요 함수를 중심으로 프로그램 개발에 대해 기술하기로 한다.

3.1 데이터 송수신 설계

데이터 전송을 수행하는 특성 상 본 논문에서 제작하려는 테스트 프로그램은 클라이언트-서버 통신 모델에 기반한다. 데이터를 송수신하는 측 각각 클라이언트, 서버에 해당하며 두 시스템은 100GbE 네트워크로 서로 연결된다. 데이터 통신은 TCP, UDP 프로토콜 모두 지원하지만 전송 안정성과 파일 신뢰성 제고를 위해 TCP를 기본 프로토콜로 설정하였다. 클라이언트는 while 루프를 돌며 매번 지정된 블록 사이즈에 해당하는 용량의 데이터를 파일로부터 읽어 네트워크로 송신하며 마지막 1바이트까지 전송될 때까지 write(int fd, void *buf size_t count)실행을 반복한다. 이는 데이터를 수신받는 서버 역시 마찬가지로서 클라이언트와 동일하게 설정된 블록 사이즈의 데이터 용량이 read(int fd, void *buf, size_t count)함수를 통해 매 while 루프마다 실행되고 버퍼가 충만하면 파일에 기록된다. 이와 같은 데이터 송수신 설계 알고리즘을 클라이언트, 서버 별로 의사코드로 나타내면 그림 3, 4와 같다.

```
Data: a file on the client
Result: transfer the file on the client to the server
initialize the file offset to zero;
While the file offset is not equal to the file size do
   read a file as the unit of block size;
   initialize buffer pointer to zero;
   while buffer pointer is not equal to the block
   size do
      write data in buffer to server
      increase the buffer pointer
   end
   increase the file offset;
end
```

그림 3. 클라이언트의 데이터 송신을 위한 의사코드
Fig. 3 Pseudo code for sending data from the client

```
Data: a file to be saved on the server
Result: Receive the file from network and save it to the
server
initialize file offset to zero
While the file offset is not equal to the file size do
   initialize the buffer pointer to zero;
   While the buffer pointer is not equal to the block size
   do
      Read data from network and write to the buffer
      increase buffer pointer
   end
   write a file as the unit of block size
   increase the file offset
end
```

그림 4. 서버의 데이터 수신을 위한 의사코드
Fig. 4 Pseudo code for receiving data from the server

위로부터 알 수 있듯이 while 문 내의 매 루프마다 파일과 메모리에 대한 입출력이 이뤄지고 네트워크를 통해 데이터가 송수신된다. 클라이언트와 서버의 원활한 파일 송수신을 위해 버퍼에 대한 입출력은 block_size를 단위 크기로 실행된다. 따라서 클라이언트는 파일로부터 읽어 메모리 버퍼에 복사된 데이터를 네트워크로 전송하고 버퍼 포인터를 증가시킨다. 반면, 네트워크 맞은편에 위치한 서버는 해당 데이터를 수신하여 메모리 버퍼에 복사하고 누적된 크기가 block_size 크기에 도달하면 디스크 상의 파일로 저장하게 된다.

3.2 데이터 송수신 주요 함수

클라이언트 - 서버 간 효율적인 데이터 전송이 이뤄지기 위해서는 비동기 방식 시스템으로 송수신이 이뤄질 필요가 있다. 이에 따라 select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout) 함수를 통해 송신 버퍼가 비워졌을 때 클라이언트가 데이터를 전송하고 서버는 데이터가 수신 버퍼에 입력되었을 때 처리되도록 하였다. 관련해서 클라이언트의 select 함수 호출을 나타내면 그림 5와 같다. 개설된 TCP 기반 소켓 cli_sock에 대한 select 함수 호출을 통해 클라이언트는 소켓 버퍼가 비워지면 write(int sd, char* buffer, size_t block_size) 함수를 통해 파일 데이터를 네트워크로 전송하고, 그렇지 않으면 대기한다. 반면 서버는 소켓의 비트가 세트되어 select함수가 양수로 반환되면 read(int sd, char* buffer, size_t block_size) 함수를 통해 해당 데이터를 서버의 버퍼로 저장한다. 파일 기

```
fd_set read_fds;
maxfdp1=getmax(cli_sock) +1;
static struct timeval timeout = 1;
while(tx_data_size < file_size ){
      FD_ZERO(&read_fds);
      FD_SET(cli_sock, &read_fds);
  ret = select(maxfdp1, &read_fds, (fd_set *)0, (fd_set
         *)0, (struct timeval *)&timeout);
      if (ret > 0){
          if(FD_ISSET(cli_sock, &read_fds)){}
          ...
           FD_CLR(cli_sock, &read_set)
      ....
}
```

그림 5. select 시스템 콜을 통한 클라이언트 상의 비동기 통신 구현
Fig. 5 Implementing asynchronous communication on the client through the select system call

술자의 경우와 달리 소켓 기술자를 통한 데이터 송수신은 네트워크를 경유하기에 block_size 용량의 데이터가 단번에 송수신되지 않고 분할되어 송수신되기도 한다. 이러한 경우 메모리에 남아있는 잔여 데이터에 대한 송수신이 구현되어야 한다. 개설된 TCP 기반 소켓 cli_sock에 대한 select 함수 호출을 통해 클라이언트는 소켓 버퍼가 비워지면 write(int sd, char* buffer, size_t block_size) 함수를 통해 파일 데이터를 네트워크로 전송하고, 그렇지 않으면 대기한다. 반면 서버는 소켓의 비트가 세트되어 select함수가 양수로 반환되면 read(int sd, char* buffer, size_t block_size) 함수를 통해 해당 데이터를 서버의 버퍼로 저장한다. 파일 기술자의 경우와 달리 소켓 기술자를 통한 데이터 송수신은 네트워크를 경유하기에 block_size 용량의 데이터가 단번에 송수신되지 않고 분할되어 송수신되기도 한다. 이러한 경우 메모리에 남아있는 잔여 데이터에 대한 송수신이 구현되어야 한다. 그림 6는 이와 관련해서 서버의 주요 코드를 간략화한 것이다. block_size로 지정된 용량의 데이터가 모두 클라이언트로부터 수신될 때까지 TCP 소켓을 이용한 통신은 계속된다. 서버의 메모리 버퍼에 파일 전송이 완료되면 while 루프는 중단되고 전송된 데이터 용량이 누적 데이터 용량을 나타내는 변수 bytes_received에 합산된다.

```
register ssize_t ret;
char *buf = recv_buffer;
register size_t nleft = block_size;

while (nleft > 0) {
      ret = read(serv_sock, buf, nleft);
      if (ret < 0) {
          if (errno == EINTR || errno == EAGAIN)
                break;
        else if (errno == ENOBUFS || errno ==
                   ENOMEM)
                return NET_SOFTERROR;
           else
                return NET_HARDERROR;
      }else if (ret == 0)
           break;
      nleft -= ret;
      buf += ret;
}
bytes_received += blksize - nleft;
bytes_received_this_interval += blksize - nleft;
```

그림 6. 클라이언트로부터 전송되는 데이터의 수신을 수행하는 코드
Fig. 6 Code that performs the reception of data sent from the client

IV. 전송 성능 향상을 위한 zero-copy 기술 적용

zero-copy는 다수의 컨텍스트 스위칭, 메모리 복사를 동반한 파일 전송에서 벗어나 파일 기술자 - 소켓 기술자 간 최단의 데이터 경로 수립을 통해 전송 효율을 향상시킨 기술이다[8]. 본 장에서는 일반 데이터 전송 프로그램에 이를 구현할 수 있는 기법에 대해 논의하기로 한다.

4.1.파일 송신 프로그램

3장에서 개발된 프로그램은 파일에서 읽어들인 데이터를 메모리 버퍼에 저장하고 이후 네트워크 상대편 서버로 전송하는 방식이다. 보다 세부적으로 데이터 처리를 살펴보면 디스크의 파일 데이터는 지정된 블록 사이즈를 단위 크기로 읽기 버퍼와 애플리케이션 버퍼, 소켓 버퍼를 거쳐 DMA 버퍼까지 총 4번에 걸쳐 복사되고 컨텍스트 스위칭 역시 동일한 횟수로 발생한다. 이로 인한 성능 한계를 극복하고 대용량 파일 전송에 활용하기 위해서는 적절한 zero-copy 함수 호출, 동적인 파일 옵셋, 목적지의 파일 저장 세 가지 사항이 고려되어져야 한다. 먼저, 첫 번째 사항의 경우 본 논문에서는 sendfile(int out_fd, int in_fd, off_t *offset, size_t count)을 호출하였다. 아래 그림 7은 zero-copy 방식의 데이터 전송을 위한 코드 일부를 보여주며 메모리 버퍼를 거치지 않고 파일에서 소켓으로 데이터 전송이 이뤄짐을 확인할 수 있다.

두 번째 고려 사항은 대용량 파일 전송 시 지속적으로 증가하는 파일 옵셋 처리에 관한 것이다. 통상 zero-copy는 소용량의 불특정 파일을 랜덤하게 전송하거나 네트워크 성능을 측정하기 위해 반복 전송하

```
register ssize_t ret;
register size_t nleft = block_size;
int ret;
static off_t offset;

while (nleft > 0) {
      ret = sendfile(cli_sock, fd, &offset, nleft);
      if (offset == file_size){
            break;
      }
      if (ret < 0) {
            switch (errno) {
                  case EINTR:
                  case EAGAIN:
                        break;
                  case ENOBUFS:
                  case ENOMEM:
                        return NET_SOFTERROR;
                  default:
                        return NET_HARDERROR;
            }
            break;
      } else if (ret == 0)
            break;
      nleft -= ret;
}
return block_size - nleft;
```

그림 7. zero-copy 구현을 위한 sendfile 함수 기반의 클라이언트 프로그램 업그레이드
Fig. 7 Upgrade of client program based on sendfile function for the implementation of zero-copy

는 용도로 활용된다. 파일 옵셋을 증가시키지 않기에 전송이 그만큼 단순하고, 안정적인 성능을 얻기가 용이하다. 하지만 대용량 파일의 경우 블록 사이즈보다 훨씬 크기 때문에 (파일 용량 / 블록 사이즈) 횟수만큼 파일로부터 데이터를 읽어 네트워크에 쓰는 작업이 반복되어야 하고 그 과정에서 파일 옵셋이 업데이트 된다. 관련해서 그림 7은 sendfile()를 이용해 개선된 파일 전송 코드로서 세 번째 인자인 off_t offset이 파일 옵셋을 나타낸다. 따라서 파일 옵셋 변수 offset이 파일 용량과 동일하면 전송은 완료되고 프로그램은 종료된다.

4.2. 파일 수신 및 저장 프로그램

데이터 송신은 sendfile() 함수를 기반으로 Zero-copy 기술을 구현할 수 있지만 데이터 수신의 경우는 지원 함수가 전무하다. 이에 따라 본 논문에서는 클라이언트로부터 수신되는 데이터를 메모리 버퍼에 단순 저장하는 그림 6의 프로그램을 확장하여 디스크에 파일로 저장 가능하도록 개선하였다. 직접적인 zero-copy 기술이 적용되지 않더라도 클라이언트로부터 수신되는 데이터를 파일로 저장하는 것이 가능하다면 유용할 것이다. 이에 따라 그림 8의 순서도에 따라 프로그램을 제작하였고 적절한 시점에서 데

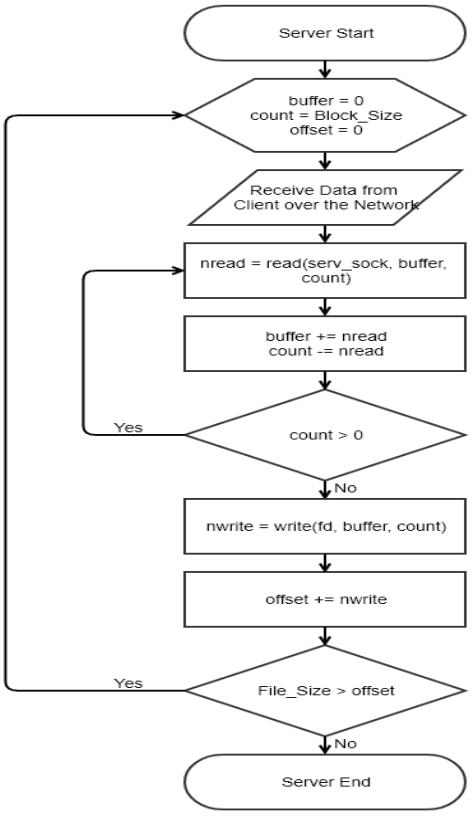

그림 8. 서버 시스템 상의 데이터 수신 및 파일 저장을 위한 동작 메커니즘
Fig. 8 Operational mechanism for receiving data and storing files on the server system

표 1. 데이터의 초고속 전송, 저장을 위한 시스템 내부 주요 컴포넌트 사양
Table 1. Specifications of major components inside the system for high-speed data transmission and storage

Components	Description
CPU	2CPU / Intel Xeon E5-2623 v3 3.0GHz,10M Cache,8.00GT/s QPI,Turbo,HT,4C/8T (105W)
Memory	64GB (4ea * 16GB) RDIMM, 2133MT/s, Dual Rank, x4 Data Width
Disk Array	16TB (18ea * 1TB) 7.2K RPM SATA 6Gbps 3.5in Hot-plug Hard Drive,13G
NIC	ConnectX®-5 VPI adapter card, EDR IB (100Gb/s) and 100GbE, single-port QSFP28, PCIe3.0 x16, tall bracket
Connect Cable	Mellanox® active fiber cable, ETH 100GbE, 100Gb/s, QSFP, LSZH, 5m

이터 수신과 파일 저장을 위한 read(server_fd, buffer, count),write(fd, buffer, count)함수를 호출하였다. 주의할 점은 서버 프로그램 실행 초기 메모리 버퍼, 데이터의 크기, 파일 옵셋에 해당하는 변수 buffer, count, offset가 각각 0, block_size, 0으로 초기화된다는 사실이다. 이후 서버의 데이터 수신 과정에서 read()함수의 두세 번째 인자인 buffer, count 변수가 각각 수신된 데이터의 크기만큼 증가 및 감소하고, count가 0일 때 메모리의 데이터를 파일로 기록한다. 이후 다시 클라이언트에서 입력되는 데이터를 void *buffer로 지정된 주소에 저장하며 앞서의 과정을 반복한다. 서버는 파일 옵셋이 파일 크기와 일치하면 파일 저장이 완료된 것으로 간주하고 서버의 데이터 수신 프로그램을 종료한다.

V. 실험 환경 구축 및 성능 평가

본 장에서는 지금까지의 과정을 통해 제작된 프로그램을 이용해 zero-copy의 유용성 및 성능을 검증하고자 한다. 이를 위해 간단한 테스트베드를 구축하였고, 다양한 조건 하에 실험을 수행하였다.

5.1 테스트베드 구성

데이터 전송, 저장을 위해 1TB 디스크 16개로 이뤄지는 두 대의 스토리지를 준비하였고 각 시스템은 송수신 역할에 따라 클라이언트, 서버로 구분된다. 실험을 위해 각 시스템에는 100GbE NIC카드를 장착하였고 5m 길이의 100GbE LSZH 케이블로 간략한 네트워크를 구성하였다. 실험을 위해 양 시스템 간 전송하고자 하는 파일의 용량은 100GB이며, 4장에서 제작된 프로그램을 통해 TCP 기반으로 전송된다. 각 시스템 및 주요 컴포넌트의 사양, 그리고 실험 환경을 표 1에 기술하였다.

네트워크를 경유한 데이터 저장에 있어 결국 병목은 말단의 디스크이다. 이는 네트워크 성능이 아무리 높아도 최종적으로 데이터가 저장되는 스토리지의 성능이 미흡하면 원하는 결과를 얻을 수 없기 때문이다. 이에 따라 디스크 성능의 한계로 인해 40Gbps 이상의 입출력은 현실적으로 불가하지만 10Gbps 이상의 데이터 전송을 위해서는 10GbE 이상의 사양이 전제되어야 하기에 100GbE 기반으로 네트워크를 구성하였다.

5.2 시스템 특성 프로파일 획득

본 논문에서는 Zero-copy를 활용했을 때 데이터

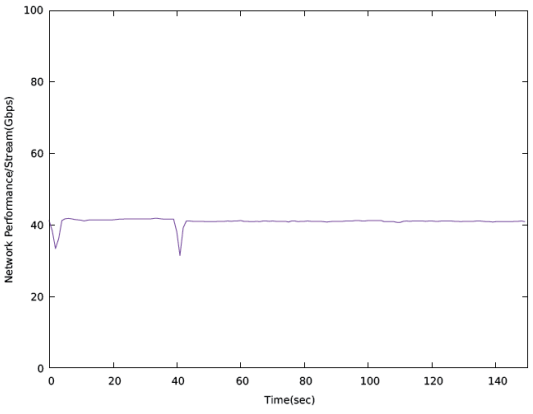

그림 9 단일 스트림 기준 클라이언트-서버 간 네트워크 성능 측정
Fig. 9 Measuring client-server network performance based on single stream

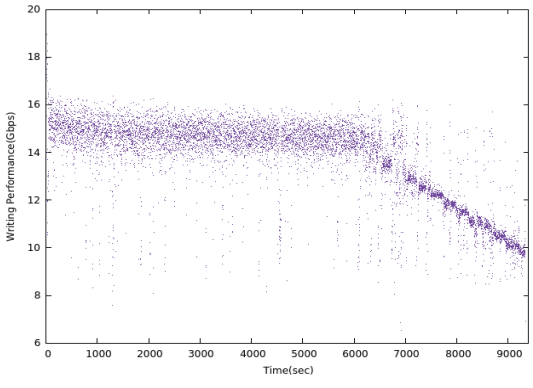

그림 10. RAID0 기반으로 생성된 16TB 스토리지의 쓰기 성능 프로파일
Fig. 10 Write performance profile of 16TB storage created based on RAID0

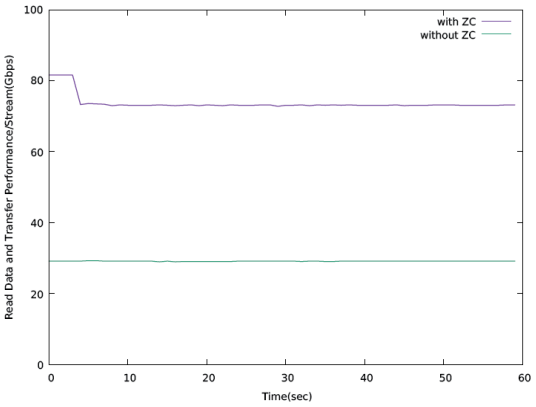

그림 11. 단일 스트림 기준 Zero-copy 사용 유무에 따른 시스템 내부의 10GB 파일 전송 성능 비교
Fig. 11 Comparison of internal 10GB file transfer performance depending on the use of zero-copy on signle stream

송신은 물론 저장의 유용성을 입증하고자 한다. 이를 위해 클라이언트의 대용량 파일을 서버로 전송할 것이고 zero-copy 적용을 통해 성능을 향상시킬 것이다. 하지만 이를 위해서는 테스트베드를 구축하는 시스템 현황을 먼저 파악하고 도달 가능한 성능을 인지하는 것이 중요하다. 관련해서 숙지해야 할 시스템 특성으로 전송, 저장 성능이 있다[9]. 먼저, 100GbE로 회선이 구성된 클라이언트-서버의 전송 성능을 측정하였고 그 결과를 나타내면 그림 9와 같다.

현재 CPU의 기술적 한계로 단일 코어에서 40Gbps를 초과하는 성능을 구현하는 것이 불가능하며, 이러한 사실은 그림 9의 그래프를 통해서도 알 수 있다. 이를 극복하기 위해 효율적인 100GbE 네트워크 활용을 위해 병렬 스트림 기법이 널리 쓰이고 있다[10], 가령, 3개의 스트림을 사용할 경우 90Gbps 이상의 성능을 효율적으로 얻을 수 있다. 하지만 CPU 코어 수에 비례하는 병렬 스트림으로 네트워크 상의 트래픽을 단순 증가시키는 것은 활용면에서는 유용할 수 있으나 본 논문의 취지에 어긋난다. 무엇보다 단일 스트림의 성능이 개선되어야 병렬 스트림의 활용성 역시 높아질 수 있다. 이에 따라 본 논문은 단일 스트림을 기본 실험 조건으로 지정하였다.

두 번째로 파악되어야 할 시스템 특성은 데이터가 기록될 스토리지의 입출력 성능이다. 테스트베드를 구성하는 각 스토리지는 1TB 디스크 16개로 구성되며 RAID0 기반으로 16TB 용량의 가상 디스크 볼륨으로 형상화했다. 이에 대한 자체 기록 성능을 평가하였고 그 결과를 그림 11에 도시하였다. 이로부터 테스트베드에 사용된 시스템의 최대 쓰기성능이 14~16Gbps이고, 외부에서 아무리 초고속으로 데이터가 수신되더라도 이 속도 이상으로 데이터를 저장하는 것이 불가능함을 알 수 있다. 이후 상세히 기술하겠지만 이러한 스토리지의 성능 한계가 병목으로 작용하고 100GbE 네트워크의 이점을 상쇄하는 결과로 나타났다.

5.3 시스템 자체 전송 성능 평가

NIC를 경유할 경우 단일 코어의 실질적 성능 한계가 40Gbps 내외라는 점은 대단히 중요하다. 시스템 내부에서 zero-copy으로 파일 전송 속도를 극대화시키더라도 네트워크를 경유해야 한다면 성능이 반감되기 때문이다. 이에 따라 시스템 버스 상에서 10GB 파일의 자체 전송 실험을 파일 옵셋, zero-copy 호출 여부에 따라 네 가지 방식으로 수행하였다.

위 그림 11은 고정된 옵셋 하에서 zero-copy 사용 유무에 따른 10GB 파일의 내부 전송 전송을 보여준다. zero-copy를 사용할 경우 초기 4초까지 80Gbps로 내부 전송되고 이후 73Gbps 내외의 성능이 안정적으로 유지된다. 반면 read, write 함수를 사용하는 기존 방식 하에서는 29Gbps로 성능이 하향되었음을 알 수 있다. 추가로 주목할 것은 파일 옵셋을 고정하여 특정

표 2. zero copy 및 파일 옵셋 조건에 따른 시스템 내부의 단일 스트림 데이터 전송 성능
Table 2. Signle stream data transfer performance inside the system according to zero copy and file offset conditions

file offset / system call	Fixed	Unfixed
zero-copy (sendfile)	73.7	51.5
non zero-copy (read-write)	29.2	24.7

부분만을 전송할 경우 zero-copy/non zero-copy 모두 전송 성능이 높게 나타났다는 사실이다. 이는 다음 표 2를 통해서도 확인할 수 있다.

파일 옵셋이 고정된 경우 zero-copy/non zero-copy 각각 각각 2.52, 2.1배의 성능 차이가 발생하였다. 이는 zero-copy 기술 적용을 통해 실질적으로 파일 전송 속도 향상을 구현할 수 있다는 사실을 보여준다. 다만, 파일 옵셋을 고정하는 방식은 파일 내 특정 부분을 반복적으로 전송하기 때문에 실질적인 파일 전송에 활용할 수 없다는 문제가 있다. 이에 따라 실질적인 전송에 초점을 맞춰 위 표 우측 칼럼의 세부 전송 내역을 나타내면 표 3과 같다.

위 결과로부터 zero-copy 방식의 경우 10GB 파일이 51Gbps 내외의 속도로 전송되어 1.66초가 소요되었음을 알 수 있다. 하지만 read, write 함수를 호출하는 기존 방식의 경우는 이보다 낮은 24Gbps 내외로 파일이 전송되었고 작업 완료까지 3.48초가 소요되었다. 이를 통해 zero-copy가 실제 파일 전송의 성능을 개선시켰음을 확인할 수 있다.

5.4 Non zero-copy 방식의 파일 전송 및 저장

보다 효율적인 데이터 전송을 위한 zero-copy의 이점을 판단하기 위해서는 기존 방식의 특성을 알고, 해당 성능을 비교 기준으로 활용할 수 있어야 한다. 이에 따라 먼저, 클라이언트에 저장되어 있는 10GB, 100GB 용량의 두 파일을 각각 zero-copy와 non zero-copy 방식으로 서버에 전송, 저장하는 일반 프로그램을 제작하였다. read, write 저수준 함수 호출을 통해 입출력이 구현되고, 해당 실험의 결과를 각각 도시하면 그림 12, 13과 같다.

두 그래프에서 빨간색과 파란색은 각각 데이터 저

표 3. 파일 옵셋 증가 시 zero-copy 사용 유무에 따른 단일 스트림 데이터 전송 성능 세부 내역
Table 3. Details of single stream data transfer performance depending on the use of zero-copy with unfixed file offset

```
## zero-copy
Interval(sec)    Transfer(GBytes)    Bandwidth(Gbps)
=====================================
0.00-1.00        5.97                51.3
1.00-1.66        4.01                51.9

## non zero-copy
Interval(sec)    Transfer(GBytes)    Bandwidth(Gbps)
=====================================
0.00-1.00        2.85                24.5
1.00-2.00        2.89                24.8
2.00-3.00        2.89                24.8
3.00-3.48        1.37                24.8
```

장, 전송 결과를 나타낸다. 먼저, 10GB 파일의 경우 13.1, 42.7Gbps로 처리되어 작업 완료까지 각각 6.54, 2.01초가 소요되었다. 100GB 처리에 있어서는 평균 10.3, 10.2Gbps로 저장, 전송되어 84초 후에 작업이 완료되었는데 이로부터 데이터의 크기가 처리 성능에 일정 부분 영향을 주었음을 알 수 있다. 또한 주목해

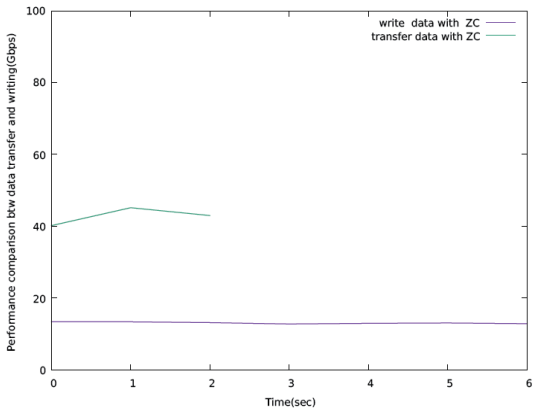

그림 12. 단일 스트림 기준 Zero-copy 기반의 10GB 파일 전송 및 저장 성능
Fig. 12 10GB file transfer and storage performance based on single stream zero-copy

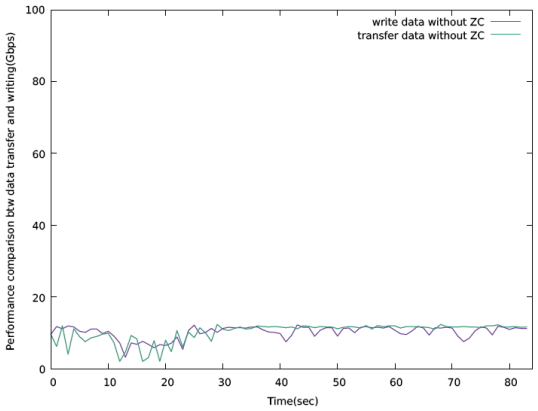

그림 13. 단일 스트림 기준 Non zero-copy 방식에서의 100GB 파일 전송 및 저장 성능 측정 결과
Fig. 13 100GB file transfer and storage performance measurement result in non zero-copy method based on single stream

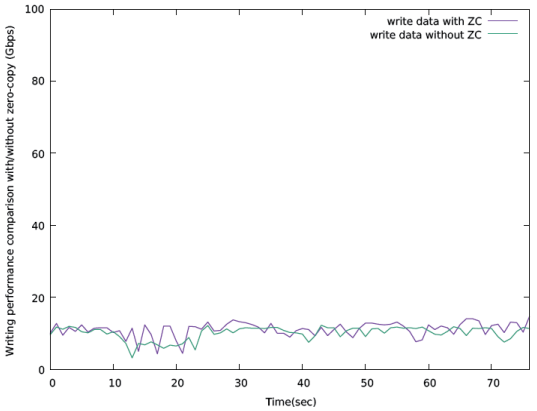

그림 14. zero-copy 사용 유무에 따른 100GB 파일 저장 성능 비교
Fig. 14 Comparison of 100GB file recording performance depending of the use of zero-copy

야 할 특이 사항으로 10GB 파일의 전송 속도와 저장 속도의 차이가 크다는 것을 들 수 있다. 그림에서도 확인할 수 있듯이 데이터 전송은 40Gbps 이상으로신속하게 완료되었지만 저장은 13.1Gbps로 3배 이상의 시간이 소요되었다. 이는 zero-copy 기술 적용을 통해 파일은 고속으로 전송되었으나 데이터 저장의 경우는 디스크가 병목으로 작용하여 처리 속도가 급격히 저하되었기 때문에 나타난 결과이다.

5.5 Zero-copy 방식의 파일 전송 및 저장

5.4절에서 100GB 용량의 파일 처리를 통해 일반적인 방식에서의 전송, 저장 성능을 대략적으로 파악하였다. 본 절에서는 zero-copy 기술을 통해 개선된 프로그램으로 동일한 실험을 수행한 결과에 대해 논의하고자 한다. 이를 위해 데이터 저장과 전송 두 가지로 분류해 접근하였고 저장 성능의 비교 결과를 그림 14에 나타내었다. 빨간색 그래프가 zero-copy 기술을 적용하였을 때의 저장 성능에 해당한다. 100GB 용량의 파일을 전송하여 서버의 스토리지에 저장하는데 76.89초가 소요되었고 평균 11.2Gbps 속도로 처리되었다. 이는 저수준 함수 호출을 통한 non zero-copy와 비교하여 시간이 7.1초 단축되고 전송 속도는 1Gbps 개선되었음을 보여준다. 하지만 앞서 지적했듯이 디스크가 성능의 병목으로 작용하기에 이것만으로 zero-copy 효과를 입증하는 것은 한계가 있다. 이에 따라 100GB 파일을 디스크 전단의 메모리까지 전달하는 실험을 추가로 진행하였고 해당 결과를 그래프로 도시하면 그림 15와 같다. 이전과 마찬가지로 빨간색 그래프와 파란색 그래프가 각각 zero-copy와 non zero-copy 방식의 성능을 보여준다. zero-copy를 이용할 경우 100GB 데이터는 평균 14.2Gbps 속도로 60.46초 동안 전송되었다. 반면 저수준 API 함수를 호

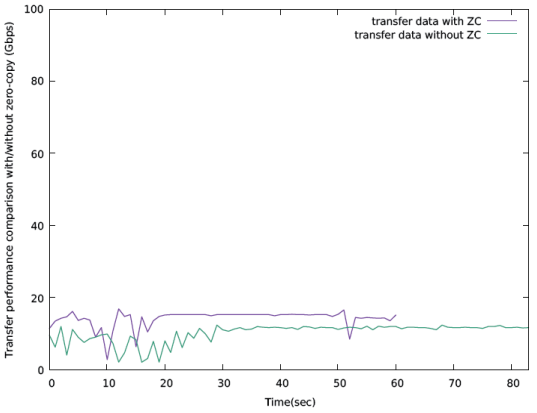

그림 15. zero-copy 사용 유무에 따른 100GB 파일 전송 성능 비교
Fig. 15 Comparison of 100GB file transfer performance depending of the use of zero-copy

출하는 non zero-copy 방식은 10.3Gbps의 평균 속도로 83.52초 후에 전송이 완료되었다. 두 방식의 차이를 전송 시간과 속도 측면에서 산출하면 각각 23.1초, 3.9Gbps로서 이전의 파일 저장 실험보다 성능 격차가 더욱 커진 것을 알 수 있다. 이는 성능 병목 지점인 디스크를 배제하였기 때문으로 풀이되며 이를 SSD, NVMe 등으로 교체하면 보다 향상된 결과를 얻을 수 있었을 것으로 예상된다.

Ⅸ. 결 론

Zero-copy는 고성능 파일 전송을 위해 리눅스 커널 2.2에서 처음 소개된 기술로서 커널 영역 - 사용자 영역 간 빈번한 컨텍스트 스위칭 및 메모리 복사를 줄여 실질적인 전송 성능을 개선하기 위해 제안되었다. 파일 용량 측면에서 zero-copy의 활용 현황을 살펴 보면 웹 애플리케이션 분야의 소규모 정적 파일이 주요 처리 대상이다. 파일 용량이 큰 경우 활용이 일반적이지 않은데, 본 논문에서는 이를 극복하기 위해 대용량 데이터셋을 다루는 분야에도 zero-copy 적용이 가능하도록 테스트 프로그램을 제작하였다. 나아가 10GB, 100GB 파일의 성능 평가 및 비교를 통해 대용량 파일의 경우에서도 zero-copy를 이용한 전송 효율 향상이 가능함을 정량적으로 입증하였다.

하지만 본 논문에서는 zero-copy 기술의 한계점 역시 분명히 확인할 수 있었는데 파일 용량 증가에 따라 전송 성능 저하, 말단의 디스크가 병목으로 작용하는 현실은 향후 개선이 필요하다. 다행히 SSD, NVMe 등 고성능 기록 장치의 보편화 속에 데이터 전송과 저장의 성능 격차가 갈수록 더욱 줄어들 것이다. 이에 따라 향후 대용량 파일 전송에도 zero-copy 기술이 일반적으로 적용되고 VLBI 등의 과학 데이터 전송 성능 개선을 위한 주요 수단으로 자리매김할 것으로 전망된다.

References

[1] D. Reinsel, J. Gantz, and J. Rydning, *The Digitization of the World From Edge to Core.* Framingham: IDC, 2018.

[2] M. Song, H. Kim, Y. Kang, D. Je, S. Wi, and S. Lee, “Implementation of Ring Buffer based Massive VLBI Data Stream Input/Output over the Wide Area Network,” *J. of the Korea Institute of Electronic Communication Sciences*, vol. 14, no. 6, 2019, pp. 1109-1120.

[3] W. Allcock, J. Bresnahan, R. Kettimuthu, and M. Link, “The Globus Striped GridFTP Framework and Server,” In *Proc. IEEE SC'05 ACM/IEEE conference on Supercomputing*, Seattle, U.S.A, 2005, pp. 54-54.

[4] Univer of Southcarolina, “NETWORK TOOLS AND PROTOCOLS Lab 9: Enhancing TCP Throughput with Parallel Streams,” *Technical report*, June 2019.

[5] S. Fuller and L. Millett, *The Future of Computing Performance: Game Over or Next Level?* Washington, D.C.: National Acamedies Press, 2011.

[6] J. Song and J. Foss, “Performance Review of Zero Copy Techniques,” *International Journal of Computer Science and Security (IJCSS).*, vol. 6, no. 4, 2012, pp. 256-268.

[7] D. Stancevic, *Zero Copy I: User-Mode Perspective.* Houston: Slashdot Media, 2003.

[8] S. Palaniappan and P. Nagaraja, *Efficient data transfer through zero copy.* New York: IBM Developer, 2008.

[9] H. Kim and M. Song, “Development and Application of HDD I/O Measurement Utility,” *J. of the Korea Institute of Electronic Communication Sciences*, vol. 15, no. 6, 2020, pp. 1151-1158.

[10] B. Tinerney, *Experiences with 40G/100G Applications.* Berkeley: ESnet, 2014.

저자 소개

송민규(Min-Gyu Song)

2001년 강원대학교 전기공학과 졸업(공학사)
2003년 강원대학교 대학원 전자공학과 졸업(공학석사)
2002년 ~현재 한국천문연구원 연구원
※ 관심분야 : 대용량 데이터 처리, 초고속 네트워크, 병렬 시스템

김효령(Hyo-Ryoung Kim)

1990년 서울대학교 천문학과 졸업(이학사)
1996년 부산대학교 대학원 천문학과 졸업(이학석사)
2003년 부산대학교 대학원 천문학과 졸업(이학박사)
1990년 ~현재 한국천문연구원 연구원
※ 관심분야 : 전파천문, 외부은하, 클러스터

강용우(Yong-Woo Kang)

1988년 부산대학교 기계설계학과 졸업(공학사)
1990년 부산대학교 대학원 지구과학과 졸업(이학석사)
2000년 부산대학교 대학원 지구과학과 졸업(이학박사)
2000년~2001년 연세대학교 박사후연구원
2002년~2006년 연세대학교 연구전임교원
2006년 ~현재 한국천문연구원 연구원
※ 관심분야 : 전파백엔드시스템, 관측천문학

제도흥(Do-Heung Je)

1992년 한양대학교 전자통신공학과 졸업(공학사)
1994년 한국과학기술원 전자전산학과 졸업(공학석사)
2002년 한국과학기술원 전자전산학과 졸업(공학박사)
2002년 ~ 현재 한국천문연구원 연구원
※ 관심분야 : 전파망원경 수신시스템 개발

위석오(Seog-Oh Wi)

1993년 전남대학교 전기공학과 공학사
1996년 전남대학교 전기공학과 공학석사
2002년 전남대학교 전기공학과 공학박사
2002년 ~현재 한국천문연구원 연구원
※ 관심분야 : KVN 시스템 유지보수 총괄, KVN 전파망원경 유지보수, 서보시스템/Hexapod 시스템 개발

이성모(Sung-Mo Lee)

1998년 인천전문대학교 통신과 졸업(학사)
2010년 ~현재 한국천문연구원 연구원
※ 관심분야 : KVN전파망원경 유지보수, 전기전자장비 관리

김승래(Seung-Rae Kim)

2006년 영동대학교 토목건설과 졸업(학사)
2010년 ~현재 한국천문연구원 연구원
※ 관심분야 : KVN 수신기 유지보수, 수신기개발 업무 지원