리눅스 커널 2.6 을 위한 Network Asynchronous I/O 의 설계와 구현

임은지[O] 김재열 차규일 안백송 정성인
한국전자통신연구원
{ejlim[O], gauri, gicha, bsahn, sijung}@etri.re.kr

Design and Implementations for Network Asynchronous I/O for Linux kernel 2.6

Eun-Ji Lim, Chei-Yul Kim, Gyu-Il Cha, Baik-Song Ahn, Sung-In Jung
Electronics and Telecommunications Research Institute

요 약

수많은 동시 사용자를 처리해야 하는 인터넷 서버에서는 다수의 연결을 효율적으로 처리 하는 것이 중요한 문제이다. 기존의 멀티 쓰레드 방식이나 이벤트 드리븐 방식이 가지는 한계를 극복하기 위한 한 가지 대안으로서 네트워크 비동기 입출력 방식을 들 수 있다. 네트워크 비동기 입출력은 입출력을 요청 한 후에 완료될 때까지 블로킹 되지 않고 즉시 다른 작업을 진행할 수 있는 방식으로서, 하나의 쓰레드에서 다중 연결을 효율적으로 처리할 수 있게 한다. 본 논문에서는 리눅스 커널에 네트워크 비동기 입출력을 구현하고 실험을 통한 성능 분석을 수행하였다.

1. 서론

웹 서버나 스트리밍 서버와 같은 대부분의 네트워크 응용들은 수많은 클라이언트로부터의 과중한 부하를 처리해야 한다. 이러한 서버들은 빈번한 요청을 처리하는 것뿐만 아니라, 수많은 동시 연결을 처리해야 한다. 따라서, 인터넷 서버에서 가장 중요한 것은 다수의 연결을 효율적으로 처리하는 것이라 하겠다. 일반적으로 다중 연결을 다루는 서비스에서는 요청과 응답이 매우 빈번하게 이루어 지는데, 서버는 이러한 요청을 빠른 시간에 받고, 최대한 효율적으로 처리하여 응답을 보낼 수 있어야 한다.

인터넷 응용 서버에서 다수의 연결에 대한 입출력을 효과적으로 다루기 위해서 몇 가지 구현 방법이 사용되어 왔다. 다중 연결을 처리하는 대표적인 방식으로 아래의 2가지 형태를 들 수 있다.

● 멀티 쓰레드 (혹은 프로세스) 방식

하나의 메인 쓰레드에서 클라이언트로부터의 연결을 기다리고 있다가 새로운 연결 소켓이 만들어 질 때마다 새로운 쓰레드를 생성한다. 쓰레드 하나당 연결 하나를 담당하도록 하여 쓰레드당 하나의 클라이언트를 처리하도록 하는 방식이다.

● 이벤트-드리븐 방식

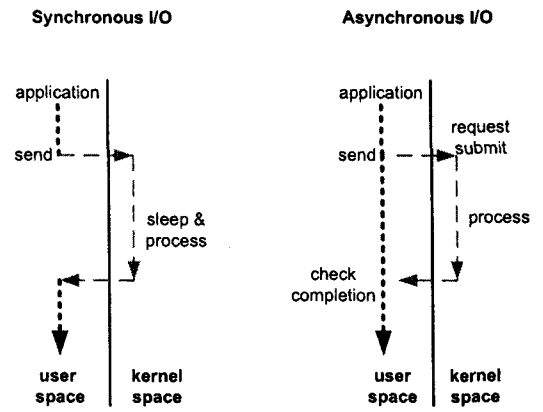
하나의 쓰레드에서 다중 연결을 처리하는 것으로서 운영체제에서 제공하는 기능을 이용하는 것이다. 운영체제는 소켓으로부터 이벤트가 발생하면 이를 응용 프로그램으로 통보하고 응용프로그램은 이벤트가 발생한 소켓을 통해서 통신 한다.

위에 설명한 두 가지 방식 중에서 멀티 쓰레드 기반의 방식은 연결당 개별적인 쓰레드를 생성하므로 서버에 동시 접속자수가 많으면 그만큼 많은 수의 쓰레드가 생성되어서 쓰레드의 문맥교환을 위한 오버헤드가 상당하게 발생할 수 있다는 단점이 있다.

이벤트-드리븐 방식은 select() 혹은 poll() 시스템 콜을 통한 폴링을 수행하여 다중 연결을 처리하는 방식이다. 운영체제는 소켓에 대한 리스트를 유지해야 하고, 리스트에 속한 각 소켓의 I/O 상태를 주기적으로 감시한다. 따라서, 최근의 연구결과에서는 수많은 다중 연결을 처리하는 고부하 서버에서 이 방식을 사용할 경우에 오버헤드가 발생하여 서버의 성능에 좋지 않은 영향을 초래할 수 있다고 밝혔다[1].

본 논문에서는 고성능 인터넷 서버에서 다중 연결을 처리하기 위한 방식으로 네트워크 비동기 입출력(네트워크 AIO)

에커니즘을 중점적으로 다룬다. AIO는 [그림 1]과 같이 쓰레드가 입출력이 완료될 때까지 블로킹 되어 기다리지 않고, 입출력을 요청한 후에 즉시 다른 작업을 진행할 수 있게 한다.

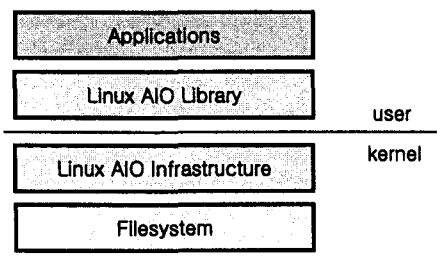


[그림 1] 동기 입출력과 비동기 입출력

현재 유닉스 계열의 운영체제에서 제공되는 소켓 인터페이스는 모두 동기적인 방식이다. 네트워크 AIO 기능을 제공하기 위해서는 운영체제의 지원이 필요하다. 현재, 리눅스 커널 2.6에 파일시스템 AIO 기능이 포함되어 있다. 즉, 파일 AIO 기능만을 지원하고 소켓 AIO 는 지원하지 않는다. 본 연구에서는 리눅스 커널 2.6.16에 포함되어 있는 파일 AIO 기능을 확장하여 네트워크 AIO를 구현하였다. 소켓에 대한 send, receive 기능뿐만 아니라, accept와 더불어 sendfile, recvfile 에 대한 AIO 기능을 지원한다. 특히, sendfile, recvfile 기능은 zero-copy 파일 전송 기능으로서 대용량의 파일을 전송할 때 유용하게 사용될 수 있다. 사용자가 네트워크 AIO를 사용하기 위해서는 기존의 파일 AIO를 사용할 때와 동일한 인터페이스를 통하여 사용할 수 있도록 하였다.

2. 리눅스 AIO

리눅스 운영체제에서 AIO를 지원하기 위한 계층 구조는 [그림 2]와 같다.



[그림 2] 리눅스 AIO 계층 구조

파일시스템의 상위에 커널 수준의 Linux AIO Infrastructure 가 존재한다. 그리고, 그 위에 사용자 계층의 라이브러리가 존재한다.

2.1. 리눅스 AIO Infrastructure

리눅스 커널 2.6은 AIO 기능을 지원 하는데, 위의 그림에서 Linux AIO Infrastructure로 나타낸 부분에 AIO 기능의 대부분이 구현되어 있다. 단, 파일에 대한 AIO 만 지원하고 있고 네트워크 소켓에 대한 AIO를 지원하지 않는다. Linux AIO Infrastructure 는 AIO 와 관련된 몇 가지 시스템 호출을 제공한다. 제공되는 시스템 호출은 다음과 같다.

- io_ setup ()

  현재 프로세스를 위한 비동기 입출력 context를 할당하고 초기화 한다.

- io_submit ()

  비동기 입출력 명령을 전달하여 입출력을 요청한다. 한 번에 다수 개의 명령을 전달할 수 있다.

- io_getevents()

  완료된 입출력 결과를 가져온다.

- io_cancel()

  비동기 입출력 명령을 취소한다.

- io_destroy()

  비동기 입출력 context를 해제한다.

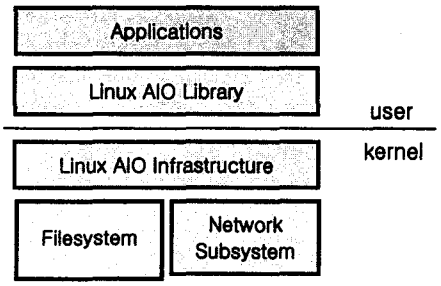
2.2. 리눅스 AIO 라이브러리

리눅스 AIO Infrastructure의 상위 계층에 리눅스 AIO 라이브러리인 libaio가 존재한다. 이것은 커널이 제공하는 시스템 콜을 사용자 수준에서 사용할 수 있도록 하는 사용자 수준의 라이브러리이다. 사용자는 libaio를 통해서 커널이 제공하는 AIO 기능을 사용할 수 있다.

3. 네트워크 AIO의 설계와 구현

3.1. 리눅스 네트워크 AIO의 구조

본 연구에서는 리눅스 커널 2.6의 파일 AIO 기능을 확장하여 네트워크 AIO를 구현한다. 네트워크 AIO를 지원하기 위한 계층 구조를 [그림 3]과 같이 설계하였다.



[그림 3] NAIO를 지원하는 리눅스 AIO 계층 구조

기존의 리눅스 AIO 를 기반으로 하여 네트워크 AIO를 지원하도록 확장하기 위해서는 AIO infrastructure 와 AIO library를 수정하여 소켓 비동기 입출력 기능을 지원하도록 해야 한다.

3.2. Linux AIO Infrastructure

3.2.1. 구조

리눅스 커널 2.6에서 네트워크 AIO를 지원하기 위한 AIO Infrastructure 의 구조는 [그림 4]와 같다. 여기서 주요한 자료구조는 다음과 같다.

- kioctx

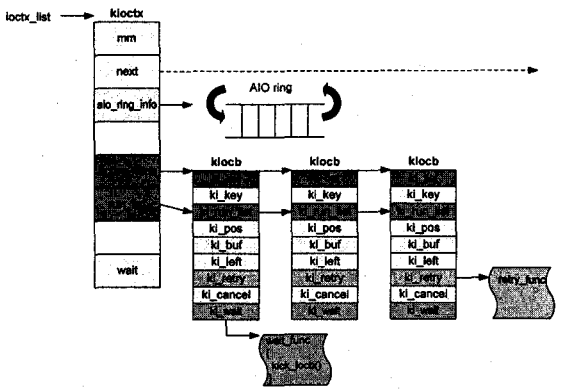
사용자 프로세스가 AIO를 이용하기 위해서는 먼저 io_setup() 시스템 호출을 통해서 AIO 컨텍스트를 생성해야 한다. 사용자 영역에서 생성된 각 AIO 컨텍스트는 kioctx라 불리는 커널의 입출력 컨텍스트와 대응된다. kioctx는 프로세스 별로 유지되며 프로세스에 의해 요청된 AIO 작업들에 대한 진행 상황 정보를 포함한다.

- kiocb

사용자가 io_submit() 시스템 호출을 통해서 비동기 입출력 작업을 요청하면 kioctx에 kiocb라 불리는 입출력 컨트롤 블록이 등록된다. Kiocb는 각각의 입출력 작업에 관련된 정보를 포함한다. 즉, 파일 기술자, 버퍼 주소, 입출력 타입 등과 같은 정보가 kiocb에 저장된다. 그리고, kiocb에는 retry 함수가 하나씩 등록된다. Retry 함수는 내부에서 블로킹되지 않는 함수로서 커널 내부에서 실제로 입출력 작업을 수행하는 부분이다. io_submit() 시스템 호출시 retry 함수가 실행되며, 블로킹되지 않고 입출력 작업을 완료 가능한 경우 그 즉시 aio_complete() 함수를 호출하여 입출력을 완료하고, 그렇지 않고 블로킹되어야 하는 경우에는 해당하는 소켓 구조체의 wait queue에 kiocb를 등록하고 바로 리턴한다. 추후에 입출력 완료 이벤트가 디바이스로부터 발생하면 wait queue에서 깨어난 kiocb 는 그 자신을 wait queue 에서 분리시켜서 다시 해당 kioctx 의 run_list 에 추가시킨다. 이후 workqueue 커널 스레드가 스케줄링 되면 kiocb 에 등록된 retry 함수를 다시 수행하게 된다. 프로세스 또는 workqueue 가 입출력 작업을 완료한 후 사용자 프로세스에게 알려주기 위해 aio_complete() 함수를 호출한다. 이 함수의 주된 역할은 kioctx 구조체에 등록되어 있는 AIO ring 에 완료된 입출력 작업의 개수만큼 io_event 구조체를 추가하는 것이다.

- AIO ring

kioctx 에는 AIO ring 이 하나씩 할당되는데, 이것은 완료된 입출력 작업의 결과가 기록되는 원형 버퍼이다. AIO ring 의 각 항목은 io_event라는 자료구조로 표현되는데, 각각은 하나의 완료된 비동기 입출력을 나타낸다. 사용자는 io_getenvents() 시스템 호출을 통해서 AIO ring으로부터 입출력 작업의 결과를 받아 온다.



[그림 4] Linux AIO Infrastructure 구조

3.2.2. 구현

사용자가 io_submit() 시스템 호출을 통해서 비동기 입출력 작업을 요청하면 요청한 비동기 입출력의 유형에 따라서 해당하는 retry 함수가 kiocb에 등록된다. AIO Infrastructure 에서 실제적인 입출력 작업이 수행되는 부분은 kiocb에 등록된 retry 함수이다. 소켓 비동기 입출력 기능을 지원하기 위해서는 소켓 비동기 입출력을 위한 retry 함수들을 구현해야 한다. 본 논문에서 소켓의 receive, send, accept, sendfile 연산을 지원하므로 각각에 해당하는 retry 함수를 구현하였다. 각 retry 함수를 aio_pread_sock, aio_pwrite_sock, aio_paccept, aio_psendfile 이라는

이름으로 정의하였다.

- **소켓 recv**

네트워크 AIO의 소켓 recv 연산을 실행하는 retry 함수는 aio_pread_sock 함수이다. aio_pread_sock 함수는 nonblocking 방식으로 BSD 소켓 오퍼레이션의 recvmsg 함수를 호출한 후, 그 반환 값을 체크한다. 만일 반환 값이 양수이면 이를 그대로 반환 하고 함수를 종료한 후 aio_complete함수를 호출하여 입출력 이벤트 구조체를 AIO ring에 등록한 후 작업을 종료한다. 만일, 반환 값이 - EAGAIN 일 경우 nonblocking 방식으로 즉시 작업을 완료할 수 없는 상황이므로 소켓의 wait queue 에 kiocb 구조체를 등록해야 한다. 그런데, 소켓 wait queue 에 등록하기 직전에 소켓에 data 가 전달되어 읽기 가능한 상태가 되는 경우를 처리하기 위하여, wait queue 에 즉시 kiocb 구조체를 등록하지 않고 wait queue에 먼저 locking 을 한다. wait queue 에 locking 을 한 후 socket의 sk_rmem_alloc 필드를 체크하여 소켓에 읽을 데이터가 준비되었는지 확인한다. 만일 이 값이 양수이면 wait queue 의 locking 을 풀고 다시 recvmsg 함수를 호출한다. 만일 sk_rmem_alloc 값이 양수가 아니면 data 를 receive 할 수 없는 상태이므로 wait queue 에 kiocb 구조체를 등록한 후 locking 을 풀고 - EIOCBRETRY 값을 반환하면서 함수를 종료한다. 소켓의 wait queue 에 등록된 kiocb 구조체는 요청한 패킷이 도착할 때 호출되는 sock_def_readable 함수에 의해서 다시 kioctx 구조체의 run_list에 등록되며, 이때 다시 retry 함수인 aio_pread_sock 함수가 호출되면서 작업을 진행하게 된다. 이러한 방식으로 작업이 완료될 때까지 wait queue 에 매달렸다가 run_list에 등록되면서 반복적으로 retry 함수를 호출하게 된다.

- **소켓 send**

네트워크 AIO의 소켓 send 연산을 실행하는 retry 함수는 aio_pwrite_sock 함수이다. aio_pwrite_sock 함수는 nonblocking 방식으로 BSD 소켓 오퍼레이션의 sendmsg 함수를 호출한 후, 그 반환 값을 체크한다. 만일 반환 값이 양수이면 나머지 데이터를 모두 보낼 때까지 반복한다. 이렇게 하여 데이터를 모두 전송하면 그 값을 반환 하고 함수를 종료한 후 aio_complete를 호출하여 입출력 이벤트 구조체를 AIO ring에 등록한 후 작업을 종료한다. 만일, 반환값이 - EAGAIN 일 경우 nonblocking 방식으로 즉시 작업을 완료할 수 없는 상황이므로 소켓의 wait queue 에 kiocb 구조체를 등록해야 한다. 그런데, 소켓 wait queue 에 등록하기 직전에 소켓이 전송 가능한 상태로 변경되는 경우를 처리하기 위하여, wait queue 에 즉시 kiocb 구조체를 등록하지 않고 wait queue에 먼저 locking 을 한다. wait queue 에 locking 을 한 후 socket의 [sk_sndbuf 필드값 - sk_wmem_queued 필드값]을 체크하여 소켓에 데이터를 전송 가능한 상태인지 확인한다. 만일 이 값이 양수이면 wait queue 의 locking 을 풀고 다시 sendmsg 함수를 호출하기 이전 부분으로 돌아간다. 만일 이 값이 양수가 아니면 data 를 send 할 수 없는 상태이므로 wait queue 에 kiocb 구조체를 등록한 후 locking 을 풀고 - EIOCBRETRY 값을 반환 하면서 함수를 종료한다. 소켓의 wait queue 에 등록된 kiocb 구조체는 소켓이 데이터를 전송 가능한 상태가 되면 호출되는 sock_def_write_space 함수에 의해서 다시 kioctx 구조체의 run_list에 등록되며, 이때 다시 retry 함수인 aio_pwrite_sock 함수가 호출되면서 작업을 진행하게 된다. 이러한 방식으로 작업이 완료될 때까지 wait queue 에 매달렸다가 run_list에 등록되면서 반복적으로 retry 함수를 호출하게 된다.

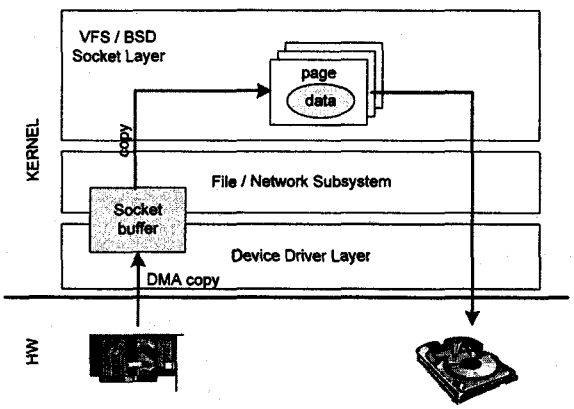
- **소켓 accept**

네트워크 AIO의 소켓 accept 연산을 실행하는 retry 함수는 aio_paccept 함수이다. 이 함수는 소켓의 accept queue 를 체크하여 만일 accept queue 가 비었으면 소켓 wait queue 에 kiocb 구조체를 등록하고 - EIOCBRETRY를 반환 한다. 만일, accept queue 가 비어있지 않다면 queue 에서 element 하나를 분리하고 새로운 connected 소켓을 초기화하여 반환 한다. 새로운 연결 소켓이 생성되면 aio_complete 함수 내에서 aio_accept_complete 함수를 호출하여 accept 결과가 사용자에게 전달될 수 있도록 kiocb 구조체와 io_event 구조체의 필드를 설정 한다. 즉, 새로운 소켓의 address를 kiocb의 aio_buf에 저장하고, addrlen을 io_event의 res2 에 저장한다. 마지막으로 sock_map_fd 함수를 실행하여 소켓과 매핑된 파일 기술자 값을 io_event 구조체의 res에 저장한다.

- **소켓 sendfile**

Sendfile 은 디스크와 TCP 소켓 간의 데이터 전송을 단일의 API로 실행되도록 하는 시스템 호출이다. 이것은 메모리 복사 횟수를 감소시켜서 성능을 향상 시킨다. 본 논문에서는 기존의 리눅스 커널에서 지원되던 sendfile 의 기능을 확장하였다. 즉, 데이터를 파일로부터 소켓으로 전송하는 경우뿐만 아니라, 소켓으로부터 파일로 전송하는 경우를 지원한다. 이 기능을 recvfile로 정의하였다. 이것은 기존의 리눅스 커널에서 지원하지 않는 기능으로서 본 논문에서 네트워크 AIO 기능으로서 recvfile 기능을 새롭게 지원한다.

사용자 계층에서 io_submit 시스템 호출을 부르면 커널에서 kiocb 구조체가 생성된다. sendfile에서 입력으로 사용될 파일 기술자는 kiocb 의 ki_filp 로 지정되고, 출력으로 사용될 파일 기술자는 kiocb 구조체의 ki_buf 로 지정된다. 비동기적 sendfile를 구현하는 kiocb의 retry함수는 aio_psendfile이다. aio_psendfile 은 입력 파일의 page cache 를 이용하여 sendfile 을 구현한다. 이 함수는 먼저 입력 파일 기술자가 소켓에 해당하는지를 체크하여 만일 소켓일 경우에 aio_recvfile 함수를 호출하고, 그렇지 않으면 aio_generic_file_sendfile 함수를 호출한다. aio_generic_file_sendfile 함수는 읽고자 하는 파일의 범위에 있는 데이터를 page cache 로 읽고 소켓의 sendpage 메소드를 통해서 소켓으로 바로 전송한다. aio_recvfile 함수는 그 반대의 기능을 수행한다. aio_recvfile()에서는 페이지를 할당하고 네트워크 소켓의 데이터를 저장하기 위한 메시지 헤더 정보를 추가하여 DMA 복사에 의해 네트워크 장치에 도착한 데이터 패킷을 메시지 내의 페이지에 저장한다. 데이터가 저장된 페이지는 commit_write()를 호출하여 디스크 페이지 쓰기 루틴에 의해 디스크 저장 장치의 파일에 저장된다. 파일의 크기만큼 전송 받은 데이터를 페이지에 저장하여 디스크에 쓰는 작업을 수행한다. 모든 데이터를 전송할 때까지 이 작업을 반복한다. [그림 5]는 recvfile의 파일 전송 개념도를 나타낸다. aio_generic_file_sendfile 또는 aio_recvfile 함수가 종료되면 반환 값을 체크한다. 만일 함수의 반환 값이 양수이고 읽을 데이터가 아직 남았으면 kiocb 구조체를 소켓의 wait queue 에 등록하고 -EIOCBRETRY 를 반환한다. 만일, 데이터를 모두 읽었으면 그 값을 반환하고 aio_complete 함수를 호출하여 입출력을 종료한다.



[그림 5] sendfile의 소켓에서 파일로의 전송 개념도

3.3. Linux AIO 라이브러리

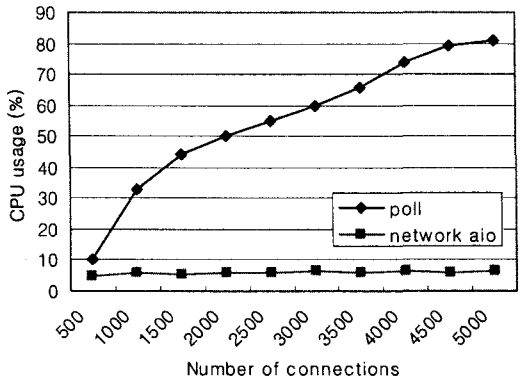
커널에 새롭게 추가된 네트워크 AIO 기능을 사용자가 사용할 수 있도록 하기 위해서는 libaio의 수정이 필요하다. 본 논문에서 네트워크 AIO 기능으로 구현한 것은 send, receive, accept, sendfile이다. 따라서 libaio 에서는 각 입출력 명령을 커널에 전달할 수 있도록 입출력 타입을 추가해야 한다. 소켓의 send, receive 에 대해서는 기존의 파일 AIO에서의 read, write 와 동일한 입출력 타입을 사용할 수 있다. 따라서 accept, sendfile에 대한 입출력 타입만 추가하였다. 아래에 libaio.h 파일의 수정된 부분을 나타내었다.

```
/* libaio.h */
...
typedef enum io_iocb_cmd {
    IO_CMD_PREAD = 0;
    IO_CMD_PWRITE = 1;
    ...
    IO_CMD_PSENDFILE = 9;
    IO_CMD_ACCEPT = 10;
}
```

4. 성능 비교 실험

Network AIO 의 성능 비교 실험을 수행하였다. 클라이언트가 서버로 메시지를 전송하면 서버는 이를 수신한 후에 동일한 메시지로 클라이언트에게 응답한다. 서버는 기존의 poll()과 Network AIO 두 버전을 구현하여 클라이언트로부터의 요청을 처리하도록 하였다. 클라이언트와 서버간의 connection 하나를 통한 request rate 를 0.2로 고정하였으며 연결된 총 connection 의 개수를 증가시키면서 서버의 CPU 사용률을 측정하였다. 이 실험은 웹서버에서 동시 사용자의 수가 증가할수록 서버의 CPU 부하가 얼마나 증가하는지를 시뮬레이션 하는 실험으로 볼 수 있다.

[그림 6]은 클라이언트와 서버간의 동시 접속 connection 의 개수를 증가했을 때의 poll()과 Network AIO 를 사용하는 서버에서 CPU 사용률을 측정한 것이다. poll()의 경우에 connection 의 수가 증가할수록 CPU 사용률이 급격하게 증가하는 반면, Network AIO 를 사용한 경우 CPU 사용률의 변화가 거의 없음을 알 수 있다. 즉, Network AIO 를 사용하는 서버에서는 수많은 동시 사용자에 대하여 high-quality 서비스를 제공할 수 있으며, 이때 서버에서 실행중인 다른 서비스의 성능저하를 초래하지 않고 서버를 운영할 수 있다.



[그림 6] poll()과 Network AIO 의 성능 비교 결과

5. 결론

본 논문에서는 다중 연결을 처리하는 인터넷 서버에서 사용할 수 있는 Network AIO 에 대해서 설명하였다. 그리고, 리눅스 커널 2.6을 위한 Network AIO의 구조와 구현 방식에 대해 기술하였으며, 실험을 통하여 기존의 이벤트 드리븐 기법과의 성능을 비교 분석하였다. 그 결과, Network AIO 를 사용한 서버는 connection의 변화에 따라서 성능이 일정한 반면, poll을 사용한 서버는 connection 이 많아질수록 서버의 CPU 사용률이 급격히 증가하는 것을 알 수 있었다. 향후, AIO sendfile 기능에 대한 성능 평가가 필요하며 Network AIO를 적용한 웹 서버에 대한 벤치마킹 테스트를 수행할 필요가 있다.

참고문헌

[1] G. Banga and J.C. Mogul. Scalable kernel performance for Internet servers under realistic loads. In Proceedings of the 1998 USENIX Annual Technical Conference, New Orleans, LA, 1998.

[2] Daniel P. Bovet et al. Understanding The Linux Kernel (3rd Edition), O'Reilly & Associates, Inc., 2006

[3] Network Asynchronous I/O for Linux, http://developer.osdl.org/bryce/naio/

[4] Kernel Asynchronous I/O (AIO) Support for Linux, http://lse.sourceforge.net/io/aio.html