유닉스 시스템에서 효율적인 CGI 게이트웨이
(An Efficient CGI Gateway in the UNIX System)

이 형 봉 †
(Hyung Bong Lee)

요 약 웹 서비스 환경에서 시시각각으로 변화하는 최신 정보나 조건에 맞는 정보를 다루기 위해서는 고정된 문서 파일 대신 실행 가능한 프로그램 파일을 지정함으로써 그 프로그램의 출력결과를 문서로 받을 수 있는 CGI 방식이 필요하다. 그런데 애초에 고안된 CGI 방식은 사용자의 요청이 있을 때마다 서비스 프로그램이 새로 생성되어야 하는 구조이기 때문에 여러 가지 성능상의 문제점을 내포하고 있다. 이를 해결하기 위해 서비스 프로그램 즉, CGI 게이트웨이의 전체 혹은 일부를 디몬 형태로 상주시키는 다양한 방안들이 제안되어 왔다. 그러나 그들 각각은 상호 배타적인 장(단점을 가지기 때문에 개발자들은 어떤 방식을 선택해야 할지를 판단할 때 혼란스럽다. 이 논문에서는 유닉스 계열의 시스템에서 게이트웨이의 주요 부분을 디몬으로 상주시키고 매번 생성되는 작은 부분으로부터 통신 채널 자체를 넘겨받는 SendFD 방식을 제안하고, 그 효용성을 실질적인 웹 환경에서 검증하였다. 제안된 방식은 기존의 소켓 파이프 방식 대비 약 3%의 성능향상을 보였다.

키워드 : CGI, FastCGI, 소켓 파이프, SendFD, HTTP

Abstract To get changing data or retrieved information in a web service environment, we need the CGI method in which we designate an executable file and receive the output of the executable file as a document. But the original CGI method has some performance pitfalls caused by the native condition that a process for the requested executable file should be created every time it is requested. Several approaches that make the process as a daemon wholly or partly have been made to solve those problems and they have exclusive strength and weakness. So, many developers are confused when they have to choose which of the CGI methods. We proposed an efficient CGI gateway design called SendFD that the main part of CGI gateway was separated as a daemon and taken over the communication channel by the other small part forked whenever it was requested. Then we testified the efficiency of SendFD styled CGI gateway in real web service environment and it showed about 3% performance improvement compared to the conventional socket pipe method.

Key words : CGI, FastCGI, socket pipe, SendFD, HTTP

1. 서 론

전형적인 WWW(World Wide Web) 환경은 그림 1과 같이 크게 웹 클라이언트(브라우저), 웹 서버, 웹 문서 파일, 게이트웨이 등 네 가지 요소로 구성된다. 웹 문서 파일은 MIME(Multipurpose Internet Mail Extensions)[1]에 정의된 형태의 문서가 고정적으로 저장된 것인데 그 중HTML로 작성된 경우가 대부분이다. 게이트웨이는 사용자가 정보를 검색하거나 실시간 정보를 요구할 때 웹 문서를 동적으로 생성하는 서비스의 입구 역할을 담당하는 실행 프로그램의 일종이다. 이는 단순히 고정적인 웹 문서를 사용하는 방법과 대비되며 주로 데이타베이스를 사용한다. 웹 서버는 인터넷을 통해 들어오는 모든 HTTP(Hyper Text Transfer Protocol)[2] 패킷을 일차적으로 식별하여, 요구된 URL (Uniform Resource Locator)의 유형에 따라 웹문서 파일 내용을 직접 참조하여 보내거나 대응되는 게이트웨이를 실행시킨 후 그 출력 결과를 수집하여 응답한다.

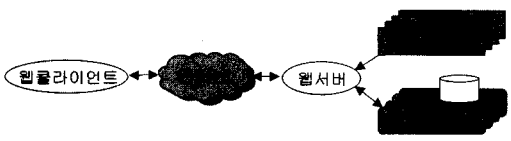

그림 1 전형적인 WWW 환경

† 종신회원 : 호남대학교 정보통신공학부 교수
hblee@honam.ac.kr
논문접수 : 2003년 8월 14일
심사완료 : 2003년 10월 27일

CGI(Common Gateway Interface)[3]는 웹 서버가 게이트웨이를 실행시킬 때 HTTP 관련 정보 및 매개변수를 전달하거나 실행 결과를 돌려 받는데 필요한 절차들의 표준이고, 애초부터 유닉스 환경을 기준으로 규격화되었다. 이 논문과 관련이 있는 CGI의 주요 골자는 아래와 같다.

- 웹 서버는 식별된 게이트웨이 프로세스를 생성(fork)한다.
- 이 때, 웹 서버는 HTTP에 관한 정보는 환경변수를 통해 전달하고, 입력 매개 변수 값은 웹 클라이언트가 지시한 CGI 메소드에 따라 GET일 경우에는 환경변수를, POST일 경우에는 표준 입력 장치(stdin, file descriptor 0)를 통해 전달한다.
- 생성된 게이트웨이는 출력 결과를 표준 출력 장치(stdout, file descriptor 1)를 통해 웹 서버에 전달한다.

위의 CGI 형태의 웹 서비스는 개발과 사용이 편리하지만, 최근 거의 모든 기업 정보시스템이 웹 정보시스템으로 전환되면서 불특정 다수의 상시 접근에 따른 성능상의 병목 지점으로 지적되어 왔다. 이와 관련된 문제점을 해결하기 위해 다양한 측면에서 다양한 방법론들이 연구되고 있는데[4-11], 그 중의 한 갈래는 게이트웨이의 빈번한 생성과 소멸에 따른 부담을 최소화시키는 분야에서 이루어지고 있다[12,13]. 이 분야는 운영체제와 DBMS와 밀접한 관계가 있기 때문에 시스템 플랫폼에 따라 접근 방향이 조금씩 다르다.

이 논문은 4장에서 웹 서비스 시험용으로 사용된 표 1의 실 시스템에서 게이트웨이의 생성에 따른 부담의 실체를 규명해 보이고, 이기용 등[13]이 제안한 응용서버 방식을 더욱 개선시킨 SendFD 방식을 제안하여 그 효용성을 단순 CGI 방식[3], 응용서버 방식, 그리고 FastCGI[12] 방식 등과 비교하여 검증한다.

2. 관련 연구 및 문제 고찰

여기서는 게이트웨이의 생성에 따른 부담을 줄이기 위한 대표적인 방안인 FastCGI와 응용서버 방식을 살펴보고, 그러한 부담의 실체를 실증적으로 분석하여 각 방법론의 접근 방향을 규명한다.

2.1 관련연구

FastCGI[12] 방식은 그림 2의 (a)와 같이 게이트웨이를 웹 서버에 등록하여 디몬으로 미리 실행시켜놓고, 요청이 있을 때마다 웹 서버가 내부 IPC(Inter Process Communication)를 통해서 게이트웨이와 직접 통신하여 처리 결과를 클라이언트에 전달한다. 이 때 IPC 기법으로 TCP/IP 혹은 유닉스 소켓을 사용한다. 이 방법에서는 게이트웨이의 생성에 따른 부담이 제거되는 대신, 웹 서버와 게이트웨이 사이의 통신 부담이 추가되고, 게이트웨이를 개발할 때 반드시 FastCGI 개발환경을 구축해야 하므로 호환성이 저하된다.

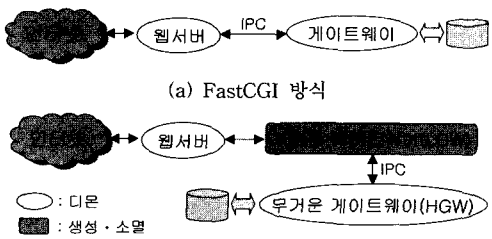

그림 2 FastCGI와 응용서버방식의 개념

응용서버 방식[13]은 그림 2의 (b)와 같이 게이트웨이를 가벼운 게이트웨이(Light Gateway, LGW)와 무거운 게이트웨이(Heavy Gateway, HGW)로 분리하여, HGW는 디몬으로 미리 생성해 두고, 웹 서버에 의해 생성된 LGW로 하여금 IPC를 통하여 HGW와 통신하여 그 처리 결과를 돌려주는 방식이다. 이 방식은 LGW를 생성하는 부담과 웹 서버~LGW 및 LGW~HGW사이의 IPC 부담을 동반하지만, CGI 본래의 표준에 충실하기 때문에 개발이 용이하고 호환성이 높다는 장점이 있다.

따라서 FastCGI 보다 더 요구되는 부담이 감수할 수 있는 정도라면 응용서버 방식도 충분한 설득력을 가진다. 이런 이유로 현재 대부분의 웹 게이트웨이는 이 방식을 따르고 있다[13].

이기용 등[13]은 이와 같은 응용서버 구조에서 특히, LGW와 HGW 사이의 IPC 방법을 메시지 큐, 보통 파일, 유닉스 소켓(소켓 파이프) 등 세 가지로 분류하여 시험한 결과, 유닉스 소켓을 가장 우수한 방법으로 결론짓고 있다.

2.2 게이트웨이 생성 부담 분석

그림 3에 그림 4의 내용을 얻기 위해 사용된 LGW(프로그램 1)와 HGW(오라클 Proc*C 프로그램)를 2000번 생성만 하는데 소요되는 총 시간(response time)과 실제 실행시간(execution time)을 보였다. HGW+DB는 HGW가 생성된 후 오라클과의 접속과 해제를 포함하도록 하였다.

그림 3으로부터 데이타베이스와 관계없는 간단한 LGW를 생성하는 부담은 거의 무시할 수 있을 정도임을 알 수 있다. 그러나 데이타베이스 접근코드를 포함하고 있는 게이트웨이는 생성하기 위한 부담도 크지만, 거기에 데이타베이스 접속시간이 고려된 부담 은 더욱 커진다는 것을 알 수 있다. 이것은 기본적으로 데이타베이스 접근코드의 규모가 클 뿐만 아니라, 오라클 자체의 클라이언트-서버 구조로 인하여 접속 시 또 다른 데이타베이스 서버를 생성해야 하기 때문이다[14].

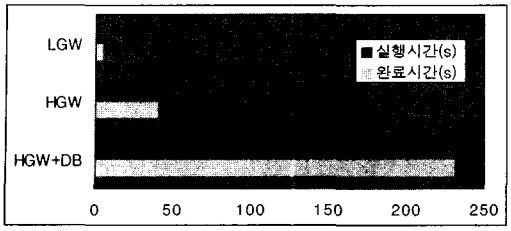

그림 3 HGW를 2000번 생성하는 부담(sec)

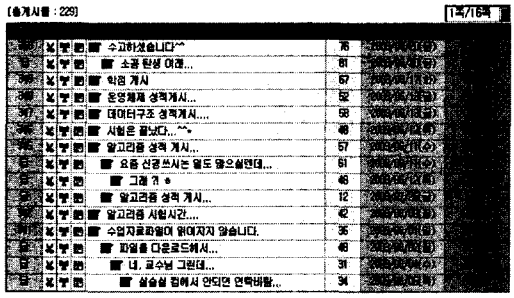

그림 4 이 논문에서 사용된 CGI 서비스 내용

FastCGI방식은 전용 웹 서버를 사용해서 게이트웨이 전체를 미리 실행시켜 놓으므로 그림 3의 세 가지 부담 모두를 절약할 수 있으며, 응용서버 방식은 HGW만 미리 생성시키므로 LGW 생성 부담만을 가진다. 그러나 LGW 생성 부담은 미미하므로 FastCGI 방식에 가까운 효과를 얻을 수 있다.

3. SendFD 방식에 의한 가벼운 게이트웨이(LGW)와 무거운 게이트웨이(HGW)의 설계와 구현

3.1 유닉스 내부의 입·출력 구조

유닉스의 모든 입·출력은 open(), create(), socket() 등 파일 개방과 관련된 시스템 콜에 의해 얻어진 정수형의 FD(File Descriptor, 파일 식별자)에 의해 이루어진다.

3.1.1 유닉스 내부의 파일개방 자료구조

파일 개방을 위한 유닉스 내부의 자료구조는 그림 5와 같다[15]. 이 그림에서 FD는 각 프로세스별로 주어지는 독립된 "u" 공간에 위치한 파일 식별자 테이블에서 할당된 엔트리 번호(array index)로 결정된다. 파일 식별자 테이블 엔트리는 모든 프로세스들이공유하는 파일 테이블 엔트리를 지시하고, 파일 테이블 엔트리는 소켓, TTY, 파이프 등 특수 구동기나 일반 파일을 위한 파일 시스템과 연결된다.

fork()가 일어나면 "u" 공간이 복사되므로 여러 프로세스가 파일 테이블을 공유하는 효과를 얻을 수 있는데, 특히 유닉스 파이프(pipe())는 이 특성을 이용한 IPC기

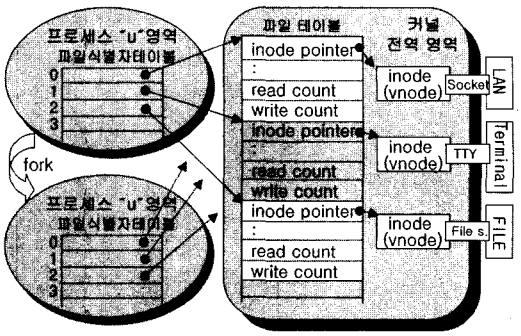

그림 5 유닉스 내부의 파일개방 자료구조

법이다. 또한 exec()에 의해 새로운 프로그램이 적재 될 때에도 "u" 공간은 그대로 유지되는데 표준 입·출력 장치는 이 특성을 기반으로 한 묵시적 표준이다. 즉, 대부분의 응용 프로그램들은 자신이 적재되기 직전 FD 0,1,2가 미리 개방된다는 사실을 전제로, 입력은 0, 출력은 1, 에러 메시지는 2 번 FD를 사용하여 입·출력을 수행한다. 이 들 표준 입·출력에는 일반 파일, 소켓, 단말기, 키보드, 프린터 등 다양한 장치들이 연결될 수 있다.

3.1.2 응용서버 방식의 게이트웨이 입·출력장치 연결

이기용 등[13]이 제안한 응용서버 방식은 위에서 언급된 FD 특성 및 소켓 연결 등에 의하여 웹 서버, LGW, HGW가 그림 6의 모습으로 연결되고, CGI 서비스의 최종 책임은 LGW에 있다. 이 그림에서 굵은 실선은 사용자 공간과 커널 공간의 경계선이고, 실선 안의 작은 원은 FD를 의미한다.

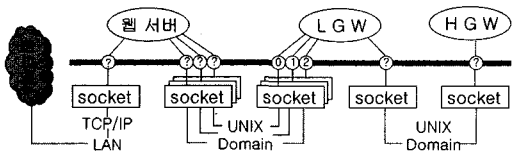

그림 6 응용서버 방식의 게이트웨이 연결 모습

3.2 SendFD 방식의 게이트웨이 설계

3.2.1 SendFD의 기능

SendFD(Send File Descriptor)란 로컬 시스템 내에서 한 프로세스가 자신이 가진 FD를 소켓 파이프를 통해서 다른 프로세스에게 전달하는 것을 말한다[16]. SendFD에 의해 FD를 전달받은 프로세스는 FD를 보내준 프로세스가 이미 개방한 입·출력 장치를 그 프로세스와 공유하는 결과를 얻는다. 그림 7에 SendFD의 개념을 보였다. 이 그림에서, 프로세스A와 프로세스B 사이에 소켓 파이프(굵은 점선)가 설정되어 있고, 개방된 입·출력 장치가 프로세스A의 FD 1에 연결된 상태다.

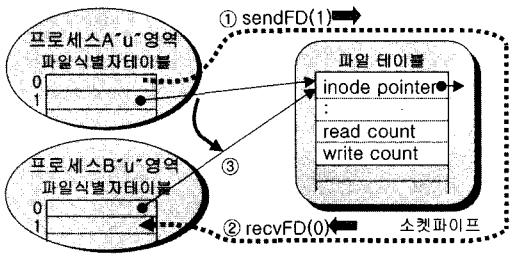

그림 7 SendFD의 개념

이 때 프로세스A가 자신이 가진 FD 1을 소켓 파이프를 통해서 보내면(①), 프로세스B는 동일한 입·출력 장치와 연결된 새로운 FD 0(운영체제가 임의로 부여함)을 받는다(②). 이는 파일 테이블을 공유하는 것이어서(③) 프로세스B는 프로세스A를 거치지 않고 곧바로 입·출력할 수 있다.

3.2.2 SendFD 방식의 게이트웨이 설계

그림 8에 SendFD를 이용한 게이트웨이의 설계 방안을 제시하고, 구현된 코드를 프로그램 1~프로그램 7에 보였다(운영체제 시스템 호출 및 여러 함수들의 에러 처리는 생략하였음).

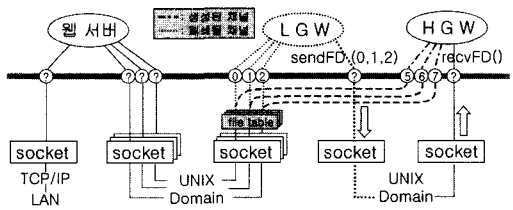

그림 8 SendFd 방식의 게이트웨이 설계

• 입·출력 FD의 전달

① HGW(프로그램 1)는 미리 디몬으로 실행되어, 특정 유닉스 도메인(경로명)에 바인드하고 연결요청을 기다린다.

② LGW(프로그램 2)는 생성되자 마자 HGW와의 약속된 소켓 파이프로의 연결을 설정하고, 웹 서버로부터 상속받은 FD 0, 1, 2을 SendFD 프리미티브인 'int sendFD(int sd, char *data, int dlen, int fd)' (프로그램 4)를 사용해서 차례차례 HGW로 보낸다.

③ HGW는 연결을 접수한 후 RecvFD 프리미티브인 'int recvFD(int sd, char *data, int dsize, int *fd)' (프로그램 5)로 LGW가 보내온 FD를 받는다. 이때 유닉스 운영체제는 HGW에게 임의의 비어있는 FD 번호(예:5, 6, 7)를 대응시켜준다.

④ LGW는 나머지 채널을 폐쇄한 후 종료한다.

⑤ HGW는 넘겨받은 FD를 그대로 사용하거나 표준 입·출력 장치인 0, 1, 2로 재설정('void mkStd(int s0, int s1, int s2)', 프로그램 6)한 후 입·출력을 수행한다.

⑥ 입·출력이 끝나면 HGW는 세 FD 모두를 폐쇄한다.

• CGI 매개변수의 전달 및 추출

LGW는 웹 서버와 연결된 입·출력 FD 이외에 CGI 표준을 따르는 몇 가지 매개 변수들을 HGW에게 넘겨주어야 한다. 이를 위하여 "매개변수1=값1\0매개변수2=값2\0…\0매개변수n=값n\0" 형태로 문자형 배열을 구성('int mkEnv(char env[])', 프로그램 3)하여 SendFD 프리미티브의 데이타 영역에 실어 보낸다.

HGW는 환경변수 대신 문자형 배열로 전달 받은 매개 변수로부터 필요한 변수의 값을 추출('char *getEnv(char env[], int elen, char *envname)', 프로그램 7)하여 사용한다.

• HGW의 결과 전송(게이트웨이 출력)

일반적으로 C 언어의 CGI 프로그램에서는 내부적으로 표준 출력장치(FD 1)를 사용하는 printf() 함수를 사용한다. SendFD 방식에서도 FD를 전달받은 후 그것을 표준 입·출력 장치로 재설정하면(FD 전달과정 ⑤), 원래의 CGI 프로그래밍 환경과 완벽한 호환성을 이룬다.

프로그램 1 LGW의 주 모듈

```
#include <sys/socket.h>
#include <sys/un.h>

main()
{
     struct  sockaddr_un  hgw_addr;
     char    env[4096];
     int     sd, elen;

     sd = socket(AF_UNIX, SOCK_STREAM, 0);

     memset(&hgw_addr, 0, sizeof(hgw_addr));
     hgw_addr.sun_family = AF_UNIX;
     strcpy(hgw_addr.sun_path, "/tmp/board");

     connect(sd, &hgw_addr, SUN_LEN(hgw_adr));

     elen = mkEnv(env);          /* 프로그램3 */

     sendFD(sd, env,  elen, 0);   /* stdin  */
     sendFD(sd, NULL, 0,    1);   /* stdout */
     sendFD(sd, NULL, 0,    2);   /* stderr */.
                                  /* 프로그램4 */
     exit(0);
}
```

프로그램 2 HGW의 주 모듈

```
#include <sys/socket.h>
#include <sys/un.h>
```

```
main()
{
      struct  sockaddr_un  hgw_addr;
      char    env[4096], qry[4096];
      int     sd, nsd, elen;

      sd = socket(AF_UNIX, SOCK_STREAM, 0);

      memset(&hgw_addr, 0, sizeof(hgw_addr));
      strcpy(hgw_addr.sun_path, "/tmp/board");
      hgw_addr.sun_family = AF_UNIX;

      bind(sd, &hgw_addr, SUN_LEN(hgw_adr));
      listen(sd, 5);

      close(0);    /* stdin  */
      close(1);    /* stdout */
      close(2);    /* stderr */

      bConnect("user", "password");
      while(1) {
          nsd = accept(sd, NULL, NULL);

          elen= recvFD(nsd, env, sizeof(env), &s0);
          (void)recvFD(nsd, NULL, 0, &s1);
          (void)recvFD(nsd, NULL, 0, &s2);
                                    /* 프로그램5 */
          close(nsd);
          mkStd(s0, s1, s2);        /* 프로그램6 */
          qp = getEnv(env, elen, "QUERY_STRING");
                                    /* 프로그램7 */
          do_service();

          close(0);
          fflush(stdout); close(1);
          fflush(stderr); close(2);
      }
      DbRelease();
}
```

프로그램 3 CGI 매개변수 배열 생성 모듈

```
#include <stdlib.h>

int
mkEnv(char env[])
{
      char  *ename[] = {
                  "REQUEST_METHOD",
                  "CONTENT_TYPE",
                  "HTTP_COOKIE",
                  "QUERY_STRING"
                   NULL
                  };
      char    *ep, *ev;
      int     i;

      for(i = 0, ep = env; ename[i]; i++) {
          if (ev = getenv(ename[i])) {
              sprintf(ep, "%s:%s", ename[i], ev);
              ep += strlen(ep) + 1;  /* 'WO' */
          }
      }
      return(ep - env);       /* total length */
}
```

프로그램 4 FD 전송 모듈

```
#include <socket.h>

int
sendFD(int sd, void *data, int dlen, int fd)
{
      struct msghdr  msg;
      struct iovec   iov[1];

      msg.msg_accrights = (caddr_t)&fd;
      msg.msg_accrightslen = sizeof(int);
      msg.msg_name = NULL;
      msg.msg_namelen = 0;

      iov[0].iov_base = data;
      iov[0].iov_len = dlen;
      msg.msg_iov = iov;
      msg.msg_iovlen = 1;

      return(sendmsg(sd, &msg, 0));
}
```

프로그램 5 FD 수신 모듈

```
#include <socket.h>

int
recvFD(int sd, void *data,int dsize, int *fd)
{
      struct msghdr    msg;
      struct iovec     iov[1];
      int     n, nfd;

      msg.msg_accrights = (caddr_t) &nfd;
      msg.msg_accrightslen = sizeof(int);
      msg.msg_name = NULL;
      msg.msg_namelen = 0;

      iov[0].iov_base = data;
      iov[0].iov_len = dsize;
      msg.msg_iov = iov;
      msg.msg_iovlen = 1;

      n = recvmsg(sd, &msg, 0);
      *fd = nfd;
      return(n);
}
```

프로그램 6 표준 입·출력 장치 재설정 모듈

```
void
mkStd(int s0, int s1, int s2)
{
      dup(s0); close(s0);   /* stdin  */
      dup(s1); close(s1);   /* stdout */
      dup(s2); close(s2);   /* stderr */
}
```

프로그램 7 CGI 매개변수 추출 모듈

```
#include <stdio.h>

char *
getEnv(char env[], int elen, char *ename)
{
     char  *ep;
     int   nl;

     for (ep = env, nl = strlen(ename);
                              ep < env + elen;  )
     {
          if (!memcmp(ep, ename, nl))
              return(ep + nl + 1); /* ':'  */
          ep += strlen(ep) + 1;     /* '₩0' */
     }
     return(NULL);
}
```

4. SendFD 방식 게이트웨이의 효용성 검증

그림 8의 SendFD 방식의 게이트웨이는 그림 6의 단순 응용서버 방식에 비하여 FD를 보내는 과정에서의 부담을 가지는 대신, HGW의 최종 출력물을 전달할 때 HGW와 LGW사이의 소켓 파이프를 통과해야 하는 부담을 제거할 수 있고, LGW가 FD 전달 후 즉시 종료할 수 있으므로 전체적인 시스템 처리 부하를 줄일 수 있다는 이점이 있다.

여기서는 이와 같은 SendFD 방식의 효용성을 검증한다.

4.1 시험 환경 및 방법

• 하드웨어 시스템 환경

표 1에 웹 서비스 성능 측정을 위한 하드웨어의 사양을 요약하였다.

• 웹 서비스 환경 및 측정 방법

이 논문에서 성능 측정을 위해 사용된 웹 서비스 환경은 오라클에 저장된 그림 4의 게시판 내용을 작성해서 출력하는 CGI 프로그램이다. 이 프로그램은 C를 호스트 언어로 하는 오라클 ESQL(Embedded SQL)로 작성하였고, 그림 4에 보인 15줄을 1페이지로 정의하고 1~5페이지의 다양한 크기로 작성할 수 있도록 하였다.

표 1 CGI 웹 서비스 시험 환경

항 목	사 양	
모 델	COMPAQ AlphaServer DS20	
C P U	Alpha Ev67 667MHz	
메 모 리	512MB	
운영체제	Digital Tru64UNIX 4.0F	
L A N	Ethernet 10M	
DBMS	Oracle 8i	
CGI 언어	Proc*C(Embedded SQL/C)	
웹 서 버	Apache Version 2.0.8	
FastCGI	Apache 서버 플러그인	Version 2.4.0
	라이브러리(SDK)	Version 1.0

성능 측정은 위의 웹 서비스를 단순 CGI 방식(SmplCGI), 단순 응용서버 방식(PipeCGI), SendFD(SfdCGI) 방식, 그리고 FastCGI 방식 등 네 가지로 구현하고, 각각의 방법에 대하여 실시한 2000번의 서비스 요구에 대한 총 응답시간(response time)과 이 때의 CPU 사용율이 고려된 처리율(through put)을 중심으로 이루어졌다.

또한, 일반적인 웹 브라우저에 의한 서비스 요구로는 요구발생 및 시간측정이 곤란하므로, HTTP 요구 패킷을 직접 보낸 후 결과를 전송받을 수 있는 요구 발생기(프로그램 8)를 사용하였다.

프로그램 8 LGW에 대한 HTTP 요구 발생기

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

main(int ac, char *av[])
{
     struct  sockaddr_in  srv_addr;
     char    buf[128], *bp, *sp;
     int     len, sd, i, n, try, count;
     char *RQ[] = {
             "Accept: image/gif, "
             "image/x-xbitmap, "
             "image/jpeg, "
             "image/pjpeg, "
             "application/vnd.ms-powerpoint, "
             "application/vnd.ms-excel, "
             "application/msword, */*₩n",
             "Accept-Language: ko₩n",
             "Accept-Encoding: gzip, deflate₩n",
             "User-Agent: Mozilla/4.0 (compatible; "
                        "MSIE 6.0; Windows 98)₩n",
             "Host: algo.honam.ac.kr₩n",
             "Connection: Keep-Alive₩n₩n",
              NULL
     };

     count = atoi(av[2]);

     for (try = 0; try < count; try++) {
         sd = socket(AF_INET, SOCK_STREAM, 0);
         memset(&srv_addr,0,sizeof(srv_addr));
         srv_addr.sin_addr.s_addr =
                        inet_addr("211.227.xx.yy");
         srv_addr.sin_family = AF_INET;
         srv_addr.sin_port = htons(80);
         connect(sd,&srv_addr,sizeof(srv_addr));

         sprintf(buf, "GET /~who/cgi-bin/"
             "SFD.cgi?SRVR_NO=%s HTTP/1.1₩n", av[1]);
```

```
        write(sd, buf, strlen(buf));
        for (i = 0; RQ[i] != NULL; i++)
            write(sd, RQ[i], strlen(RQ[i]));

        shutdown(sd, SHUT_WR);
        for (len = 0; ; len += n) {
            if (read(sd, buf, sizeof(buf)) <= 0)
                break;
        }

        close(sd);
    }
    exit(0);
}
```

4.2 성능 측정 및 분석

4.2.1 반응시간(response time)

그림 9에 웹 요구 발생기 프로그램 8을 사용해서 그림 4의 웹 페이지를 쉬지 않고 2000번 내려 받는데 소요된 반응시간을 네 가지 CGI 방법론에 대하여 측정한 결과를 보였다. 이 그림에서 단순 CGI(Smpl CGI) 방법은 2장에서 측정한 대로 게이트웨이 전체의 생성과 소멸에 따른 부담으로 인하여 다른 방법론에 비하여 반응시간 성능이 크게 낮음을 볼 수 있다.

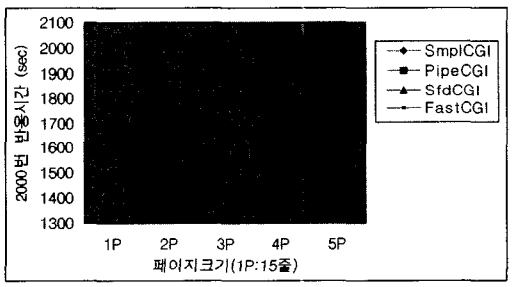

그림 9 프로그램 8에 의한 반응시간

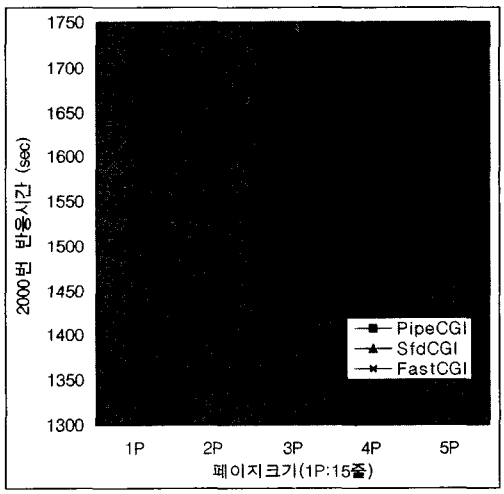

그림 10 그림 9의 세 가지 근접 부분의 확대

그러나 FastCGI 방식(그림 2의 a), 소켓 파이프를 이용한 단순 응용서버 방식(그림 2의 b, PipeCGI), 그리고 SendFD 방식(그림 8, SfdCGI)은 근소한 차이를 보이고 있다.

그림 10은 그림 9에서 근접한 세 가지 방법론 부분을 분리하여 확대한 그림으로 PipeCGI를 기준으로 하여 나머지 두 방식의 상대적인 반응시간의 백분율을 기입하였다. 이 그림으로부터 반응시간 측면에서 SendFD 방식이 이기용 등[13]이 제안한 단순 응용서버 방식(소켓 파이프 방식)보다 우수하다는 결론을 얻을 수 있다.

FastCGI 방식은 전용 웹 서버 모듈을 사용하므로 범용 웹 서버의 CGI 표준에 충실한 다른 두 방법보다 당연히 우수할 수 밖에 없다.

4.2.2 처리율(throughput)

그림 11에는 그림 9에 대한 CPU 사용율을 나타내었다. 이 그림에서 SmplCGI의 경우 다른 방법론에 비하여 CPU 사용율이 월등히 높기 때문에 이를 고려하면 종합적인 성능은 더욱 악화된다. 즉, 동일한 CPU 사용율이라면 그림 9에 보인 SmplCGI의 상대적 반응시간은 더욱 느려진다는 것이다.

그림 9에 나타낸 2000번 요구에 대한 반응시간을 처리율(초당 처리 건수)로 계산한 후, 여기에 그림 11을 적용하여 동일한CPU 처리 능력 한계(100%) 하에서의 처리율로 변환(아래 식 (1) 참조)하면 각 방법론들 간의 절대적인 비교가 가능하다.

$$p_{100} = (n \div t) \times (100 \div u) \qquad \text{식 (1)}$$

단, t : 측정된 반응시간(sec)

n : 처리건수(2000)

u : 측정된 CPU 사용율(%)

그림 12는 그림 9와 그림 11에 식 (1)을 적용하여 CPU 사용률 100% 하에서의 처리율을 나타낸 것이다. 이 그림으로부터 처리율 또한 반응시간과 마찬가지의

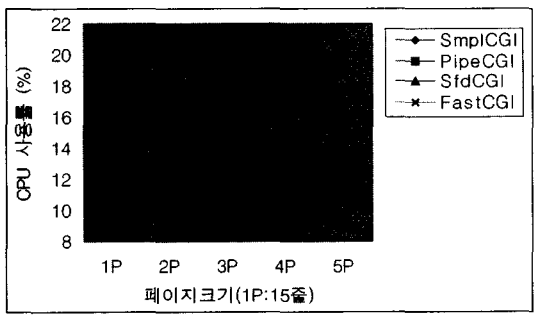

그림 11 그림 9에 대한 CPU 사용률

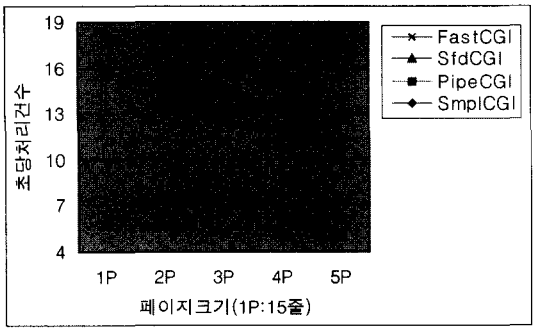

그림 12 CPU 사용률 100% 하에서의 처리율

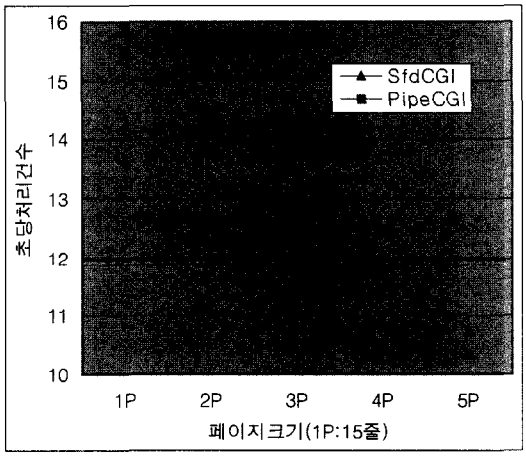

그림 13 그림 12에서 근접한 부분의 확대

성능추이를 보이고 있음을 볼 수 있다. 그림 13은 이 논문의 주 목적인 PipeCGI와 SfdCGI의 비교를 확대하여, PipeCGI를 기준으로 한 SfdCGI방식의 상대적인 처리율의 백분율을 표시한 것인데 SfdCGI 방식의 우수함을 보이고 있다.

4.2.3 종합 비교

표 2에 네 가지 CGI 게이트웨이 설계 기법을 종합적으로 비교하였다. 이 표로부터 전용 웹 서버나 웹 서버 모듈을 사용하지 않는 경우 SendFD 방식이 가장 우수하다는 결론을 얻을 수 있다.

표 2 CGI 게이트웨이 설계 기법의 종합 비교

방법론 / 비교항목	Smpl CGI	Pipe CGI	Sfd CGI	Fast CGI
반응시간 순위 (PipeCGI 대비%)	4 (119.4)	3 (100)	**2 (99.0)**	1 (98.2)
처리율 순위 (PipeCGI 대비%)	4 (42.3)	3 (100)	**2 (103.0)**	1 (114.6)
웹 서버	범용	범용	**범용**	전용
CGI 호환성	완벽	완벽	**완벽**	미흡

5. 결 론

관공서나 정보검색 등 불특정 다수가 접속하는 웹 사이트는 상시 과부하 상태에 놓인 경우가 자주 있다. 상시 과부하는 아니더라도 최소한 하루 중 과부하 시간을 맞이하는 경우는 매우 빈번하다. 이럴 경우 웹 서비스의 신뢰도는 치명적으로 저하된다. 이를 방지하기 위하여 시스템 증설 등 다양한 방법들이 있을 수 있으나, 이 논문에서는 CGI환경에서 게이트웨이의 성능향상을 위한 설계 방법인 SendFD 방식을 제안하고 그 효용성을 검증하였다.

SendFD 방식이 기존에 무의식적으로 사용돼 왔던 소켓 파이프에 의한 단순 응용서버 방식에 비하여 반응시간 및 처리율 모두 우수함을 확인하였고, 프로그래밍 기법 또한 단순 CGI 방식과 완벽하게 호환되므로 프로그램의 생산성 또한 뛰어남을 보였다. 일반적으로 반응시간과 처리율이 상호배타적(trade off)이라는 점을 고려하면 SendFD 방식의 의의는 더욱 크다.

SendFd 방식이 FastCGI 방식보다는 성능이 떨어지지만, FastCGI 방식이 전용 웹 서버나 웹 서버 모듈을 사용해야 한다는 등의 단점이 있기 때문에, 범용 웹 서버 환경에서는 SendFD 방식이 가장 우수하다는 결론을 내릴 수 있다.

이번 연구는 게이트웨이의 설계 방법에 관한 비교적 좁은 영역에서 이루어졌지만, 앞으로 DBMS의 특성, 운영체제의 특성, 네트워크의 특성이 고려된 최선의 게이트웨이 프로그래밍 기법에 관한 연구가 이어질 것이다.

참 고 문 헌

[1] RFC1521, RFC2522, *"MIME(Multipurpose Internet Mail Extensions),"* ftp://ftp.rfc-editor.org/in-notes/rfc1521.txt.

[2] World Wide Web Consortium, *"HTTP-Hypertext Transfer Protocol,"* http://www.w3.org/protocols/.

[3] World Wide Web Consortium, *"CGI: Common Gate way Interface,"* http://www.w3.org/hypertext/wwwCGI.

[4] A. Iyengar, E. Nahum, A. Shaikh and R. Tewari, *"Enhancing Web Performance,"* IPIP World Computer Congress, Montreal, Canada, 2002.

[5] E. Nahum, T. Barzilai and D. Kandlur, *"Perfo rmance Issues in WWW Servers,"* IEEE/ACM Transactions on Networking, 10(2):2-11, 2002.

[6] P. Joubert, Robert B. King, Rich Neves and Mark Russinovich, John M. Tracey, *"High-Perfo rmance Memory-Based Web Servers: Kernel and*

User-space Performance," Proceedings of the USENIX Annual Technical Conference, Boston, MA, 2001.

[7] A. Iyengar, J. Challenger, D. Dias and P. Dantzig, *"High-Performance Web Site Design Techniques,"* IEEE Internet Computing, 4(2), 2000.

[8] 정진국, 낭종호, 박성용, "다중 처리기 기반 웹 서버 구조의 실험적 성능 분석", 정보과학회 논문지: 정보통신 제28권 제1호, pp.22~36, 2001.

[9] M.N. Mills III, L.Krueger, W.Chiu, N.Halim, J.L.Hellerstein and M.S.Squillante, *"Metrics for Performance Tuing of Web-Based Appli cations,"* The computer Measurement Group, 2000.

[10] 김수정, 백승구, 김종근, "웹 정보시스템의 서비스 성능 향상을 위한 부하균형 모델 제안", 정보처리학회 논문지 제6권 제11호, pp.3179~3189, 1999.

[11] James C. Hu, S. Mungee and D. C. Schmidt, "Techniques for Developing and Measuring High Performance Web servers over High Speed Networks," Inforcom'98, 1998.

[12] Open Market, "FastCGI," http://www.fastcgi.com.

[13] 이기용, 곽태영, 서정현, 김명호, "대규모 온라인 검색 요구를 효율적으로 처리하기 위한 KRISTAL-II 웹 게이트웨이의 설계 및 구현", 정보과학회논문지:컴퓨팅의 실제 제6권 제5호, pp.496~504, 2000.

[14] Oracle Education Part, *Oracle8 Administra tion Volume 1*, Oracle Press, p.1~8, 1998.

[15] U. Vahalia, *UNIX Internals-the new frontiers*, Prentice Hall, p.241, 1996.

[16] W. R. Steven, *Advanced Programming in the UNIX® Environment*, Addison Wesley, p.479, 1992.

이 형 봉

1984년 서울대학교 계산통계학과 이학사 1986년 서울대학교 대학원 계산통계학과 (전산과학) 이학석사. 2002년 강원대학교 대학원 컴퓨터과학과 이학박사. 1986년~1994년 LG전자 컴퓨터 연구소 선임연구원. 1994년~1999년 한국디지탈이큅먼트㈜ 책임. 1997년~1999년 전자계산조직응용,전자계산기,정보통신 기술사. 1999년~현재 호남대학교 정보통신공학부 조교수. 관심분야는 프로그램 언어 및 보안, 운영체제, 멀티미디어 통신