# 유닉스 시스템에서 C 언어 출력 방법이 CGI 게이트웨이 성능에 미치는 영향

이 형 봉[†] · 정 연 철[††] · 권 기 현[†††]


## 요 약

CGI는 유닉스 운영체제의 표준 입·출력 환경에서 프로그램의 출력 결과가 고정적인 웹 문서를 대신하도록 고안된 게이트웨이와 웹 서버 사이의 표준 접속 규약이다. 따라서 CGI 게이트웨이에서는 사용된 언어가 제공하는 표준 입·출력 문장을 사용하는 것이 자연스럽다. 그런데 표준 입·출력 메커니즘은 보편적인 환경에 적합하도록 운영체제에 투명하게 설계된 버퍼 전략 중의 하나이다. 이것은 CGI 환경이라는 독특한 특성이 고려될 경우 표준 입·출력 부분이 웹 성능향상을 위한 또 다른 최적화 대상이 될 수 있음을 의미한다. 이 논문에서는 유닉스/리눅스 시스템에서 C 언어로 작성된 CGI 게이트웨이를 위한 출력의 최적화 분야를 표준 출력 방법과 파일 출력 방법으로 분류하고, 각 분야별 제안된 최적화 방안들을 Debian LINUX, IBM AIX, SUN Solaris, Digital UNIX 등 네 운영체제를 대상으로 적용하여 그 영향을 실행시간 위주로 분석하였다. 그 결과 운영체제에 따라 상당한 차이를 보였는데, 기본 방법에 비해 표준 출력 분야에서 10% 이상 향상된 경우가 있었던 반면 성능 향상이 당연시 되었던 파일 출력 방법에서는 오히려 60% 이상 저하되는 최악의 경우가 관찰되었다.


# The Effect of C Language Output Method to the Performance of CGI Gateway in the UNIX Systems

Hyung-Bong Lee[†] · Yeon-Chul Jeong[††] · Ki-Hyeon Kweon[†††]


## ABSTRACT

CGI is a standard interface rule between web server and gateway devised for the gateway's standard output to replace a static web document in UNIX environment. So, it is common to use standard I/O statements provided by the programming language for the CGI gateway. But the standard I/O mechanism is one of buffer strategies that are designed transparently to operating system and optimized for generic cases. This means that it may be useful to apply another optimization to the standard I/O environment in CGI gateway. In this paper, we introduced standard output method and file output method as the two output optimization areas for CGI gateways written in C language in the UNIX/LINUX systems, and applied the proposed methods of each area to Debian LINUX, IBM AIX, SUN Solaris, Digital UNIX respectively. Then we analyzed the effect of them focused on execution time. The results were different from operating system to operating system. Compared to normal situation, the best case of standard output area showed about 10% improvement and the worst case showed 60% degradation in file output area where some performance improvements were expected.


**키워드 : CGI 게이트웨이(CGI Gateway), 표준 출력 버퍼정책(Standard Output Buffer Discipline), 표준 출력 버퍼영역(Domain of Standard Output Buffer), 파일 출력 버퍼(File Output Buffer)**

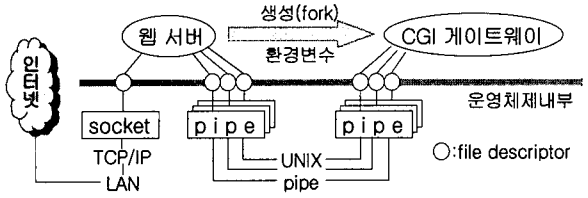
## 1. 서 론

최근 기업, 관공서, 학교 등에서 거의 모든 정보시스템이 웹 환경으로 구축되고 있다. 넓은 의미에서 웹 환경은 클라이언트-서버 모델로 분류될 수 있으나, 클라이언트와 서버가 HTTP 표준 프로토콜[1]을 사용함으로써 클라이언트가 특정 제품이나 기술에 종속되지 않는다는 점에서 전통적인 클라이언트-서버 모델[2]과 구분된다.

초창기 웹의 주요 목적은 정보의 공유나 공개에 있었기 때문에 HTML로 작성된 고정된 문서 내용을 파일에 저장해 놓았다가, 사용자의 요청이 있을 때 단순히 해당 파일을 읽어서 주고 받는 형태의 정적인 것이었다. 그러나 이런 방식은 시시각각으로 변화하는 최신 정보 혹은 조건에 맞는 정보를 제공하거나, 사용자와 관리자가 실시간으로 입력하는 비즈니스 자료를 다룰 수 없다는 한계를 가진다. CGI (Common Gate way Interface) 방식은 그러한 한계를 극복하기 위해서, (그림 1)과 같이 고정된 문서 파일 대신에 실

† 종신회원 : 강릉대학교 컴퓨터공학과 조교수
†† 정 회 원 : 호남대학교 컴퓨터게임학과 조교수
††† 정 회 원 : 삼척대학교 정보통신공학과 조교수
논문접수 : 2004년 6월 9일, 심사완료 : 2004년 11월 18일

행 가능한 프로그램(CGI 게이트웨이) 파일을 지정하여 입력 매개변수를 전달한 후, 그 프로그램의 실행 결과 즉, 출력물이 문서를 대신할 수 있도록 고안된 웹 서버와 프로그램 사이의 표준 접속 규약이다[3].




( 그림 1 ) 전형적인 CGI 환경

CGI는 애초부터 유닉스 환경을 배경으로 하고 있는데 그 가운데 이 논문과 관련이 있는 주요 내용은 아래와 같다.

**• CGI 게이트웨이의 생성**

웹 서버는 고정된 문서와 마찬가지로 CGI 프로그램이 요구될 때마다 해당 게이트웨이를 매번 새로 적재하여 생성한다.

**• 파일 식별자**

모든 입·출력은 운영체제에 대한 개방 요청인 open(), create(), pipe(), socket () 등에 의해 얻어지는 정수 식별자를 사용하여 이루어 진다. 이렇게 얻어진 파일 식별자는 프로그램 작성자의 의지에 따라 dup(), fcntl() 등의 운영체제 요청을 통해 특정 번호로 변경될 수 있다.

**• 파일 식별자의 유전**

모든 프로세스는 부모 프로세스의 fork() 요청에 의해 생성되고, 이 때 입·출력을 위한 파일 식별자는 특별한 지시가 없는 한 그대로 유전된다. 따라서 새로 생성된 CGI 게이트웨이는 웹 서버가 개방한 파일 식별자들을 그대로 상속 받는다.

**• 표준 입·출력 파일 식별자**

표준 입·출력 식별자란 각각의 독립된 프로그램이 처리 자료를 입력할 파일 식별자, 처리 결과를 출력할 파일 식별자, 그리고 오류 정보 등 긴급 메시지를 출력할 파일 식별자 등 세 개의 파일 식별자를 위해 묵시적 약속으로 예약된 번호 즉, 0, 1, 2번 식별자를 말한다. 이들 파일 식별자들은 해당 프로세스를 생성하는 부모 프로세스의 의지에 따라 키보드나 단말기, 네트워크, 파이프 등 다양한 입·출력 장치 혹은 파일이 개방되어 설정된다. CGI 게이트웨이의 경우 (그림 1)과 같이 웹 서버가 주로 파이프 장치를 개방하여 설정한 파일 식별자 0, 1, 2가 부여된 상태로 생성된다. 따라서, CGI 게이트웨이로의 표준 입력은 웹 서버가 파이프를 통해 전달하는 내용이 되고, 게이트웨이로부터의 표준 출력은 파이프를 통해서 웹 서버에 전달된다.

**• 표준 입·출력 라이브러리**

표준 입·출력 라이브러리는 표준 입·출력 파일 식별자 0, 1, 2가 개방되어 있다는 점을 전제로 입·출력 지시자를 생략하거나, 표준 입·출력 지시자인 stdin, stdout, stderr를 사용하는 gets(), printf(), fgets(), fprintf() 등의 라이브러리를 말한다. 일반적으로 C 언어로 작성되는 CGI 게이트웨이는 표준 입·출력 라이브러리를 사용해서 매개변수를 받거나 웹 문서를 출력한다.

표준 입·출력 라이브러리는 운영체제에 투명하게 사용될 수 있도록 설계된 보편적인 버퍼 전략 중의 하나이기 때문에 CGI 환경에서 웹 서비스 성능 향상을 위한 또 다른 최적화 목표가 될 수 있다. 즉, 표준 입·출력 부분은 시스템이 기본으로 제공하는 운영체제의 일부라는 인식 하에 관심의 대상에서 벗어나 있었으나, 상용화된 각 운영체제의 특성에 따라서는 표준 라이브러리를 사용하지 않았을 때 더 높은 성능을 얻을 수도 있다는 것이다. 반대로 CGI 게이트웨이를 이용한 파일 서비스의 경우, 단순히 파일 내용을 읽어서 표준 장치로 출력하는 과정을 반복하는 전통적인 방법을, 파일 내용을 읽어내지 않고 운영체제 내에서 직접 출력 장치로 전달하는 방법으로 개선하면 메모리 복사 과정이 줄어 성능이 향상된다는 사실이 알려져 있는데[15], 이 방법은 운영체제의 구현 특성에 따라 그 효과가 달라질 수 있다.

이 논문에서는 CGI 게이트웨이의 출력 부문을 웹 문서 작성을 위한 표준 출력과 파일 서비스를 위한 파일 출력으로 분류하여 각 분야별 개선 방안들을 모색하고, 그 방안들을 Debian LINUX, SUN Solaris, IBM AIX, Digital UNIX 등 네 운영체제에 적용하여 각 운영체제별로 가장 적합한 개선 방안을 제시한다. 이를 위하여 2장에서 관련 연구, 유닉스의 표준 입·출력 메커니즘과 운영체제별 차이점, 그리고 시스템 영역 버퍼와 파일의 직접 출력을 위한MMF메커니즘을 살펴보고, 3장에서 CGI 게이트웨이의 출력 성능을 개선할 수 있는 방안들을 모색하며, 4장에서 제안된 개선 방안들을 각 운영체제에 동일하게 적용하여 측정된 성능을 분석한 후, 5장에서 그 결과를 토대로 각 운영체제별 가장 적합한 개선 방안을 제시하고 검증한다. 그리고 마지막 6장의 결론으로 이 논문을 맺는다.

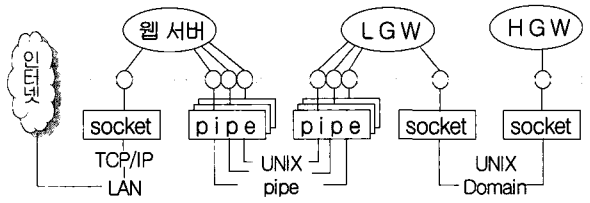
## 2. 관련 연구, 유닉스 표준 입·출력 및 MMF 메커니즘

### 2.1 관련 연구

웹 서비스 성능 향상을 위한 시도는 캐싱, 부하 분산, 웹 서버 구조, 네트워크 프로토콜 및 응용프로그램 개선 등 크게 네 분야로 대별해 볼 수 있다. 캐싱은 클라이언트나 서버에 웹 문서를 캐시로 관리하여 네트워크나 입·출력에서의 부담을 줄이려는 시도로서 가장 오래된 연구분야에 속한다[4-6]. 부하 분산 분야에서는 웹 서버 시스템을 물리적으로 중복시킨 후, 집중된 부하를 효율적으로 분산시키기 위한 연구가 이루어지고[6-9], 웹 서버 구조 분야에서는 특정 시스템이나 환경 하에서 최적의 웹 서버 구조 및 구현에 관한 연구가 이루어진다[10-13]. 프로토콜 및 어플리케

이션 개선 분야에서는 웹 서비스 요청의 특성을 고려한 TCP/IP 프로토콜 최적화 및 웹 어플리케이션을 지원하기 위한 운영체제의 기능 보완[14, 15]이나 웹 어플리케이션 자체의 효율적인 구조[16-18]에 관한 연구가 이루어진다.

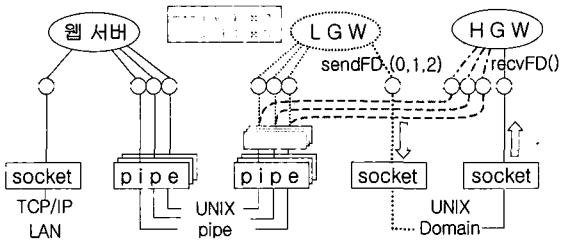
특히 [16-18]은 CGI 환경에서 효율적인 게이트웨이 구현 방안에 관한 연구로서 본 논문과 관련이 깊다. [16]은 CGI 프로그램의 빈번한 생성에 따른 부담을 제거하기 하기 위해 웹 서버를 변형하여 해당 프로그램을 영구히 등록하는 데몬 형태의 서버로 동작할 수 있도록 하였고, [17]은 CGI 프로그램의 표준을 엄격히 유지하면서도 프로세스 생성에 따른 부담을 경감시키기 위해 (그림 2)와 같이 CGI 프로그램을 가벼운 부분(Light Gateway, LGW)과 무거운 부분(Heavy Gateway, HGW)으로 분리하여 HGW은 데몬으로 동작시키고 가벼운 부분만을 매번 생성시키는 응용 서버 방식에서, LGW와 HGW 사이의 가장 효과적인 통신 방안을 제안하였다.




(그림 2) 응용 서버 방식

[18]은 [17]에서 제안한 LGW와 HGW 사이의 통신 수단을 두는 대신에, (그림 3)과 같이 LGW가 웹 서버로부터 상속 받은 표준 입·출력 파일 식별자를 HGW에게 전달하여 HGW로 하여금 해당 파일 식별자들을 직접 접근할 수 있도록 함으로써 웹 서비스 성능을 향상시킨 SendFD 방식을 제안하였다.

본 논문은, [16-18]이 CGI 게이트웨이의 구조 및 내부 통신 방법 개선에 의한 성능 향상을 도모한 데 비하여, 유닉스 운영체제마다 독특하게 동작하는 표준 출력 방법을 최적화하여 성능 향상을 얻는 방안에 대하여 논한다.




(그림 3) SendFD 방식

### 2.2 유닉스 표준 입·출력 메커니즘

#### 2.2.1 표준 입·출력의 기능 및 필요성

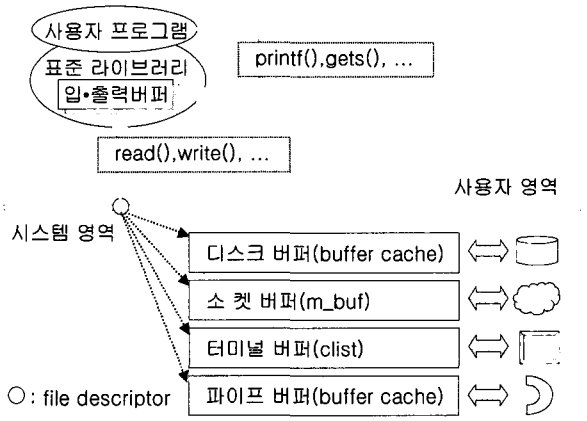
유닉스의 표준 입·출력은 (그림 4)와 같이 사용자가 작성한 응용프로그램과 운영체제 사이에서 버퍼기능을 제공하는 것으로 그 주된 목적은 다음과 같다[19].

**● 시스템 호출 빈도 감소에 의한 성능향상**

사용자 프로그램이 소량의 데이터를 자주 입·출력할 때는 해당 데이터를 운영체제에게 매번 즉시 전달하는 것 보다는 일정량이 될 때까지 임시 보관하였다가 한꺼번에 전달하는 것이 성능상 유리하다. 그 이유는 운영체제에 진입하기 위한 시스템 호출에 따른 부담이 크기 때문이다.

**● 다양한 출력 정책(discipline)의 적용**

출력장치와 사용자 프로그램이 출력하는 데이터의 성질에 따라 독특한 정책을 적용할 수 있다. 예를 들어 단말장치는 보통 줄 단위의 데이터가 출력되었을 때 의미가 있으므로 한 줄의 데이터가 완성될 때까지 출력을 보류하는 정책이 효율적일 수 있다.




(그림 4) 유닉스 표준 입·출력 메커니즘

#### 2.2.2 표준 출력 버퍼정책의 유형 및 설정

**● 표준 출력 버퍼정책의 유형**

표준 출력 버퍼정책이란 표준 출력 라이브러리를 통해 출력되는 데이터를 언제 운영체제에 전달할 것인가를 말하는데, <표 1>과 같이 세 가지 가능한 정책 유형이 있다[20].

<표 1> 유닉스 표준 출력 버퍼정책의 유형

| 정책유형 | 정책내용 |
|---|---|
| 완전 버퍼 정책 (_IOFBF) | (그림 3)의 표준 라이브러리가 관리하는 출력 버퍼의 크기 만큼 계속 저장하였다가 버퍼가 넘치면 비로서 버퍼 내용 전체를 운영체제에 한꺼번에 전달한다. |
| 라인 버퍼 정책 (_IOLBF) | 하나의 라인이 완성되거나 버퍼가 넘칠 때까지 저장하였다가 해당 내용을 운영체제에 전달한다. |
| 비 버퍼 정책 (_IONBF) | 버퍼를 사용하지 않고 표준 라이브러리에 전달된 모든 데이터를 곧바로 운영체제에 전달한다. |

**● 표준 출력 버퍼의 기본 설정**

표준 출력 라이브러리는 버퍼가 최초로 참조될 때 <표 2>에 주어진 세 가지 항목을 각각 기본(default)으로 설정한다[20].

〈표 2〉 표준 라이브러리의 입·출력 버퍼 기본 설정

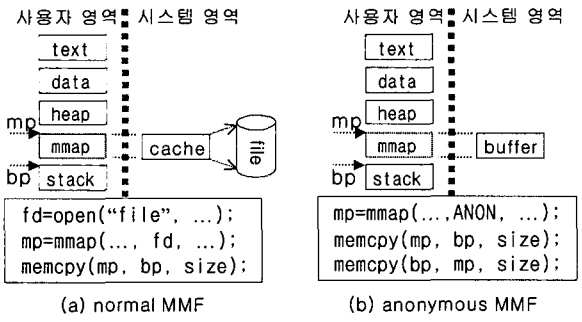
| 설정항목 | 기 본 설 정 |
|---|---|
| 버퍼 정책 | 개방된 장치의 유형에 따라 완전 버퍼 정책이나 라인 버퍼 정책 중 하나를 선택하여 적용한다. 즉, 터미널이나 유사 터미널 장치인 경우는 라인 버퍼 정책을 적용하고, 그 외에는 모두 완전 버퍼정책을 적용한다. |
| 버퍼 할당 | malloc()을 사용하여 힙 영역의 버퍼를 할당한다. |
| 버퍼 크기 | 할당되는 버퍼의 크기로는 시스템 설치 시 결정된 값(stdio.h에 정의된 BUFSIZ)을 사용한다. |

**• 표준 출력 버퍼의 임의 설정**

프로그램 작성자는 최초의 데이터가 출력되기 직전 setbuf(), setvbuf(), setbuffer() 등을 사용해서 <표 2>의 설정 항목들을 명시적으로 지정하거나 변경할 수 있다[20].

### 2.3 유닉스 MMF메커니즘

MMF(Memory Mapped File)[21]는 원래 (그림 5)의 (a)와 같이 디스크 파일을 사용자 영역 주소공간에 대응시켜 read(), write() 등 입·출력 시스템 호출 대신 주소 참조만으로 파일 입·출력 효과를 얻는 목적으로 제안되었다. 이 경우 여러 프로세스에 의한 파일 공유가 용이해지는 부수적인 장점도 얻을 수 있다. 그러나 최근에는 그 활용 범위가 넓어지면서 (그림 5)의 (b)와 같이 오로지 시스템 영역의 메모리(버퍼)를 할당하는 용도 즉, anonymous MMF로 자주 사용된다.



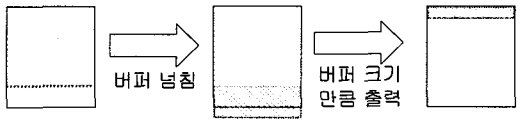

( 그림 5 ) MMF의 개념

## 3. 표준 출력 버퍼와 MMF가 CGI 게이트웨이 성능에 미치는 영향

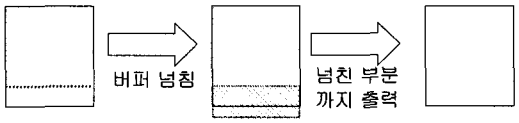
전형적인 CGI 프로그램의 출력은 표준 출력 문장을 통해서 이루어지므로 <표 2>에 나열된 표준 출력 버퍼에 대한 설정 항목들은 모두 CGI 게이트웨이의 성능에 영향을 미칠 수 있다.

### 3.1 버퍼 정책의 영향

(그림 1)에 나타난 바와 같이 CGI 게이트웨이의 표준 출력은 웹 서버와 연결된 파이프 장치로 설정되어 있다. 이 경우 라인 버퍼 정책이나 비 버퍼 정책은 완전 버퍼 정책에 비하여 시스템 호출에 따른 부담이 클 수밖에 없다. 따라서 표준 라이브러리가 애초에 기본으로 설정하는 완전 버퍼 정책이 최선의 선택이다. 그런데 완전 버퍼 정책을 적용할 경우 (그림 6)과 같이 두 가지 변형이 가능하다. (a) 방식에서는 버퍼가 꽉 차는 순간 버퍼 내용만 출력되고 나머지 자투리 출력 부분은 버퍼에 저장된다. (b) 방식에서는 버퍼 내용과 함께 넘친 자투리 부분까지 출력한다. 이 경우 넘친 부분을 버퍼에 연속해서 저장할 수 없기 때문에 두 번의 시스템 호출이 필요하다.




(a) 정확히 버퍼 크기 만큼만 출력




(b) 버퍼와 넘친 부분 모두를 출력

( 그림 6 ) 완전 버퍼 정책의 두 변형

### 3.2 버퍼 할당의 영향

버퍼 할당과 관련된 요소는 버퍼 크기와 버퍼 영역 등 두 가지 측면에서 살펴볼 수 있다.

#### 3.2.1 버퍼 크기

버퍼의 크기는 연결된 출력 장치의 특성과 시스템 호출 부담에 영향을 줄 수 있다. (그림 1)에서 CGI 게이트웨이의 표준 출력이 파이프 장치에 연결되어 있으므로 버퍼의 크기는 가능한 한 파이프 장치의 버퍼 단위와 일치하는 것이 좋고, 버퍼의 크기가 너무 적으면 시스템 호출 부담이 커지며, 너무 크면 얻어지는 성능 이득이 반감될 것임을 예측할 수 있다.

#### 3.2.2 버퍼 영역

표준 출력 버퍼는 기본으로 사용자 영역인 힙 영역에 할당되지만 사용자는 MMF를 사용해서 시스템 영역에 할당할 수도 있다.

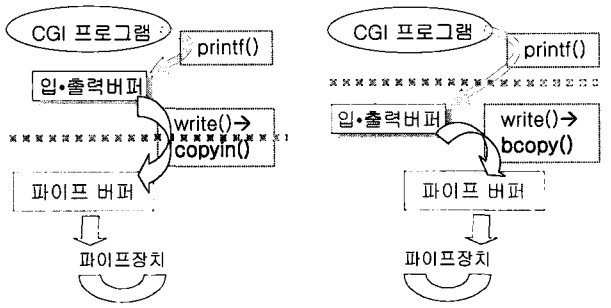
MMF에 의한 시스템 영역의 버퍼는 아래의 두 가지 측면에서 CGI 프로그램의 성능에 영향을 미칠 수 있다.

**• 표준 출력 버퍼로 사용된 경우**

(그림 7)의 (a)처럼 출력 버퍼가 사용자 영역에 존재할 경우, 버퍼 전체가 출력될 때 그 내용이 copyin()에 의한 영역간 복사가 이루어져야 하는 부담이 있다.

그러나 (그림 7)의 (b)와 같이 MMF에 의해 출력 버퍼가

시스템 영역에 위치할 경우에는 라이브러리가 표준 출력 버퍼를 접근할 때 MMF 부담이 따르는 반면, 버퍼 내용이 출력될 때에는 영역간 복사가 필요치 않다. 따라서 이 두 가지 사이의 우열은 영역간 복사(copyin(), copyout())와 MMF의 성능에 좌우된다.



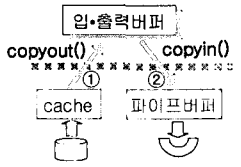

(a) 사용자 영역 입•출력 버퍼 (b) 시스템 영역 입•출력 버퍼

( 그림 7 ) 표준 출력 버퍼영역이 성능에 미치는 영향

**• 파일 서비스에서 버퍼로 사용된 경우**

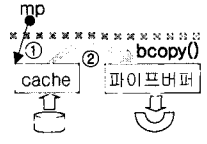
단순 문서나 이미지, 혹은 멀티미디어 등의 고정된 문서를 웹 환경에서 내려 받을 때 서버 측 동작은 매우 단순하다. 전통적인 방법인 (그림 8)의 (a)는 시스템 영역의 파일 내용을 사용자 영역의 입·출력 버퍼로 읽어낸 후 다시 시스템 영역의 파이프 버퍼로 복사한다. 이 경우 영역간에 두 번의 복사가 필요하다. 반면에 (그림 8)의 (b)와 같이 MMF를 이용하여 시스템 영역에 위치하는 파일 자체를 버퍼로 이용하면 한 번의 내부 복사로 충분하다.

```
fd = open("file", ...);
while (1) {
  n = read(fd, buf, size);①
  if (n <= 0) break;
  write(fileno(stdout),buf,n);②
}
```




(a) 전통적인 파일 서비스

```
mp = mmap("file", ...); ①
for(cp=mp; cp<mp+size; cp+=BUFSIZ)
{
  write(fileno(stdout),cp,BUFSIZ);②
}
```




(b) MMF를 이용한 파일 서비스

( 그림 8 ) 버퍼영역이 파일 서비스 성능에 미치는 영향

### 3.3 개선 가능한 출력 방법

(그림 6), (그림 7), (그림 8)로부터 성능 향상이 기대되는 여러 가지 특성 인자들을 출력 분야별로 조합하여 제안한 출력 방법들을 <표 3>에 요약하였다. 여기서 여러 가지 방법들을 제안한 이유는 이들 각 방법들이 모든 운영체제에서 동일한 효과를 발휘하지 않을 수 있기 때문이다.

<표 3> 제안된 표준 출력 및 파일 출력 방법들

| | |
|---|---|
| 표준 출력 방법 | ① <표 2>의 기본 설정을 그대로 사용함(default) |
| | ② (그림 7)의 (b)와 같이 시스템 영역 버퍼를 설정함 |
| | ③ (그림 6)의 방법 (a)를 (그림 9)의 (a)와 같이 구현하여 출력 라이브러리로 사용함 |
| | ④ (그림 6)의 방법 (b)를 (그림 9)의 (b)와 같이 구현하여 출력 라이브러리로 사용함 |
| | ⑤ ②+③ |
| | ⑥ ②+④ |
| 파일 출력 방법 | ⑦ (그림 8)의 방법 (a)에 의한 파일 서비스 |
| | ⑧ (그림 8)의 방법 (b)에 의한 파일 서비스 |

```
static char pbuf[PBSIZ+128], *cp = pbuf, ep = pbuf+PBSIZ;
Printf(char *fmt, char *a1,
       char *a2, char *a3)
{
  int   n;
  n = sprintf(cp,a1,a2,a3);
  if (cp + n < ep)
    cp += n;
  else {
    write(1, pbuf, PBSIZ);
    n = (cp +n) - ep;
    memcpy(pbuf, ep, n);
    cp = pbuf + n;
}
```

(a) 방법_(a)

```
Printf(char *fmt, char *a1,
       char *a2, char *a3)
{
  int   n;
  n = sprintf(cp,a1,a2,a3);
  if (cp + n < ep)
    cp += n;
  else {
    n = (cp +n) - ep;
    write(1, pbuf,
          PBSIZ +n);
    cp = pbuf;
}
```

(b) 방법_(b)

( 그림 9 ) (그림 6)의 방법 (a)와 (b)의 구현

## 4. 운영체제별 성능 측정 및 분석

### 4.1 성능 지표 및 시험 환경

#### 4.1.1 성능 지표

이 논문에서 관심을 두고 있는 성능 지표는 (그림 1)의 CGI 게이트웨이가 <표 3>에 제안된 각각의 출력 방법에 따라 <표 4>에 나열된 여러 운영체제 상에서 일정량의 게시판 데이터(①~⑥)나 파일 내용(⑦, ⑧)이 파이프를 통해서 웹 서버까지 전달되는 데 소요되는 사용자 모드 및 시스템 모드에서의 실행 시간이다. 그 실행 시간 결과는 각 운영체제 별 최적의 출력 방법 결정을 위한 기준으로 적용한다.

#### 4.1.2 성능 시험 시스템 환경

<표 3>의 방법들에 따라 표준 출력 및 파일 출력 성능을 측정한 시험용 시스템들을 <표 4>에 요약하였고, <표 5>에는 이들 시스템에 탑재된 운영체제 각각의 기본 설정값을 보였다.

〈표 4〉 성능 측정 대상 시스템

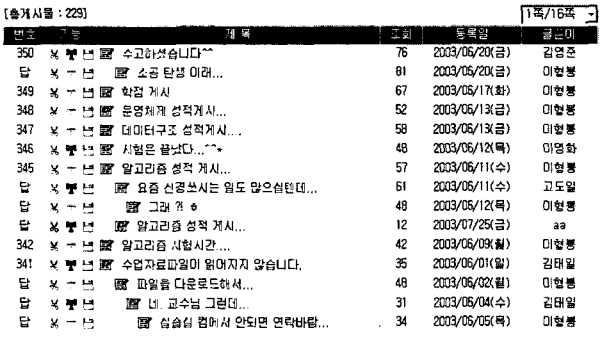
| 플 랫 폼 | 운 영 체 제 | 프 로 세 서 | 메모리 |
|---|---|---|---|
| LG-IBM PC 8478 | Debian LINUX 2.4.18 | Pentium III 1GHz | 756MB |
| IBM RS/6000 WS140 | IBM AIX 4.3 | PowerPC 604e 233MHz | 64M |
| SUN Enter-prise 250 | SUN Solaris 5.7 | UltraSparc-II, 300MHz | 128MB |
| DEC Alpha Server DS20 | Digital Tru64 UNIX 4.0F | Alpha EV67 500MHz | 512MB |

〈표 5〉 〈표 4〉의 운영체제별 기본 설정 값

| 운 영 체 제 | BUFSIZ | 그림 5 특성 (b) | 그림 6 특성 | 파이프 버퍼크기 |
|---|---|---|---|---|
| Debian LINUX | 8K | 가능 | (a) | 4K |
| IBM AIX | 4K | 가능 | (b) | 32K |
| SUN Solaris | 1K | 불가 | (a) | 64K |
| Digital UNIX | 8K | 가능 | (b) | 64K |

4.1.3 성능 시험 소프트웨어 환경

CGI 게이트웨이는 <표 3>에 분류된 각각의 방법에 따라 ①~⑥의 경우에는 파일에 저장된 (그림 10)의 게시 글 5페이지를 읽어서 2000회 출력하고, ⑦, ⑧의 경우에는 1K~3M 크기의 텍스트 파일을 1000회(IBM AIX와 SUN Solaris에서는 시스템 성능 관계로 300회로 설정하였음) 출력하여 웹 서버에 전달되도록 작성하였다. 이 때 게시 글 내용은 아래의 DB질의 인터페이스를 그대로 흉내 낼 수 있도록 파일에 구성하여 <표 4>의 모든 운영체제에서 동일한 조건으로 사용될 수 있도록 하였다.


[총게시물 : 229] 1쪽/16쪽

| 번호 | 기능 | 제목 | 조회 | 등록일 | 글쓴이 |
|---|---|---|---|---|---|
| 350 | 보기 변경 | 수고하셨습니다^^ | 76 | 2003/06/20(금) | 김영준 |
| 답 | 보기 변경 | 소중 탄생 이래... | 81 | 2003/06/20(금) | 이형봉 |
| 349 | 보기 변경 | 학점 게시 | 67 | 2003/06/17(화) | 이형봉 |
| 348 | 보기 변경 | 운영체제 성적게시... | 52 | 2003/06/13(금) | 이형봉 |
| 347 | 보기 변경 | 데이터구조 성적게시... | 58 | 2003/06/13(금) | 이형봉 |
| 346 | 보기 변경 | 시험은 끝났다...^^* | 48 | 2003/06/12(목) | 이영하 |
| 345 | 보기 변경 | 알고리즘 성적 게시... | 57 | 2003/06/11(수) | 이형봉 |
| 답 | 보기 변경 | 요즘 신경쓰시는 일도 많으실텐데... | 61 | 2003/06/11(수) | 고도일 |
| 답 | 보기 변경 | 그래 게 ㅎ | 48 | 2003/06/12(목) | 이형봉 |
| 답 | 보기 변경 | 알고리즘 성적 게시... | 12 | 2003/07/25(금) | aa |
| 342 | 보기 변경 | 알고리즘 시험시간... | 42 | 2003/06/09(월) | 이형봉 |
| 341 | 보기 변경 | 수업자료파일이 읽어지지 않습니다. | 35 | 2003/06/01(일) | 김태일 |
| 답 | 보기 변경 | 파일을 다운로드해서... | 48 | 2003/06/02(월) | 이형봉 |
| 답 | 보기 변경 | 네. 교수님 그럼데... | 31 | 2003/06/04(수) | 김태일 |
| 답 | 보기 변경 | 실습실 컴에서 안되면 연락바람... | 34 | 2003/06/05(목) | 이형봉 |

( 그림 10 ) 성능 측정에 사용된 게시판 내용

- DbConnect(user/passwd) : DB에 접속한다.
- GetCount() : DB에 저장된 게시 글의 총 개수를 얻는다.
- GetLine(n) : n=0이면 게시 글에 대한 커서(cursor)를 개방하고, n=-1이면 커서를 폐쇄한다. n>0이면 현재 개방된 커서로부터 n-1개의 게시 글을 건너 뛴 후 다음 글 하나를 읽는다.
- DbRelease() : DB 접속을 해제한다.

또한 (그림 1)에 보인 상용 웹 서버를 그대로 사용할 경우 CGI 게이트웨이의 정확한 실행 시간 측정이 곤란하므로 아래와 같이 동작하는 웹 서버 스터브를 고안하여 대신 사용하였다.

- 먼저 CGI 게이트웨이가 유닉스 도메인(AF_UNIX) 소켓에 대기(listen)한다.
- 웹 서버 스터브가 CGI 게이트웨이와 유닉스 도메인 연결을 설정(connect)한 후, 파이프를 생성하여 그 중 쓰기 가능한 파일 식별자를 연결된 소켓을 통해서 CGI 게이트웨이에 전달한다[18](웹 서버 스터브가 게이트웨이를 직접 생성(fork)하면 게이트웨이에 대한 정확한 실행 시간 측정이 곤란함).
- CGI 게이트웨이는 게시판 내용을 작성하여 전달 받은 파이프에 출력한다.
- 웹 서버 스터브는 파이프의 읽기 파일 식별자로부터 CGI 게이트웨이의 출력 내용을 전달 받아 그대로 버린다.
- CGI 게이트웨이가 출력을 마친 후 파이프를 닫고 종료하면 웹 서버 스터브도 종료한다.

4.2 성능 측정 및 분석

4.2.1 표준 출력 방법에 따른 운영체제별 성능

<표 3>에서 분류한 표준 출력 방법 ①~⑥에 따라 버퍼 크기를 변화시키면서 실시한 각 운영체제별 성능 측정 결과는 (그림 11)과 같다. 이 그림을 기본 출력 방법인 ①과 기본 버퍼크기 범례인 '-■-'를 기준으로 분석하면 아래와 같다.

- **Debian LINUX((그림 11)의 (a))**

표준 출력 버퍼의 영역에 따른(②) 성능 차이는 거의 없고, 그 밖의 방법들은 오히려 성능향상에 악영향을 미치며, 버퍼 크기의 확장이 성능 향상에 도움이 된다. 즉, 방법 ①에서 버퍼 크기를 8K에서 16K로 변경하면 약 2.4%의 성능 향상이 이루어진다.

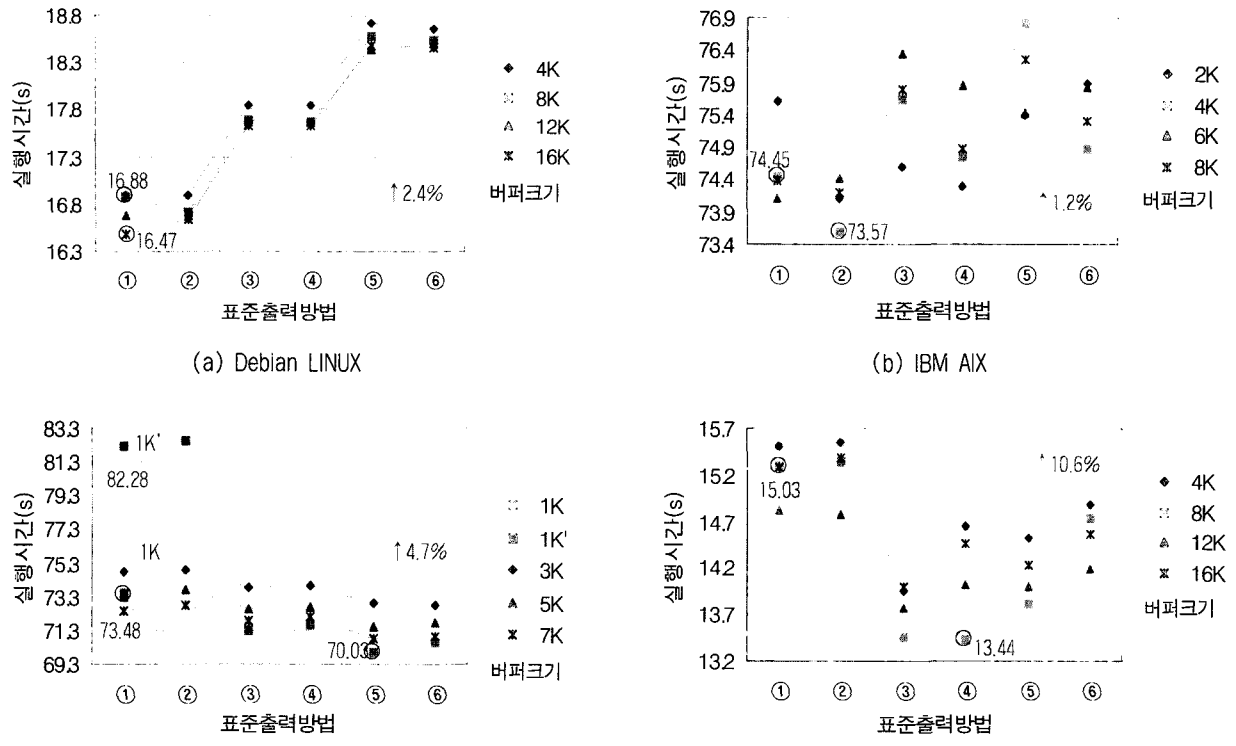
- **IBM AIX((그림 11)의 (b))**

출력 버퍼를 시스템 영역에 설정하는 방법 ②가 가장 우수하고, 버퍼 크기에 따른 변화는 각각의 방법에 따라 일정치 않는데 방법 ②에서는 기본 설정 값인 4K에서 가장 큰 성능 향상이 이루어진다. 이 경우 얻어진 성능 이득은 약 1.2%이다.

- **SUN Solaris((그림 11)의 (c))**

기본 버퍼 크기인 1K에서, 같은 크기의 버퍼를 setbuf() 등을 통해서 명시적으로 설정해 주는 경우(1K')와 그렇지 않고 라이브러리 내부 할당에 맡기는 경우(1K)의 두 성능에서 큰 차이를 보였는데 그 이유는 밝혀지지 않았다. 각각의 방법에 있어서 대체적으로 버퍼 크기의 확장이 성능 향상에 도움이 되는 경향이 있지만 유독 ③~⑥에서는 1K가 가장 적합하고, 시스템 영역의 버퍼가 사용자 영역보다 우수하다.

가장 우수한 방법인 ⑤에서 1K 버퍼를 사용할 경우 버퍼 조작 없이 기본 출력 라이브러리를 그대로 사용하는 경우에 비해 약 4.0% 정도의 성능 향상이 이루어진다.




(a) Debian LINUX

(b) IBM AIX

(c) SUN Solaris

(d) Digital UNIX

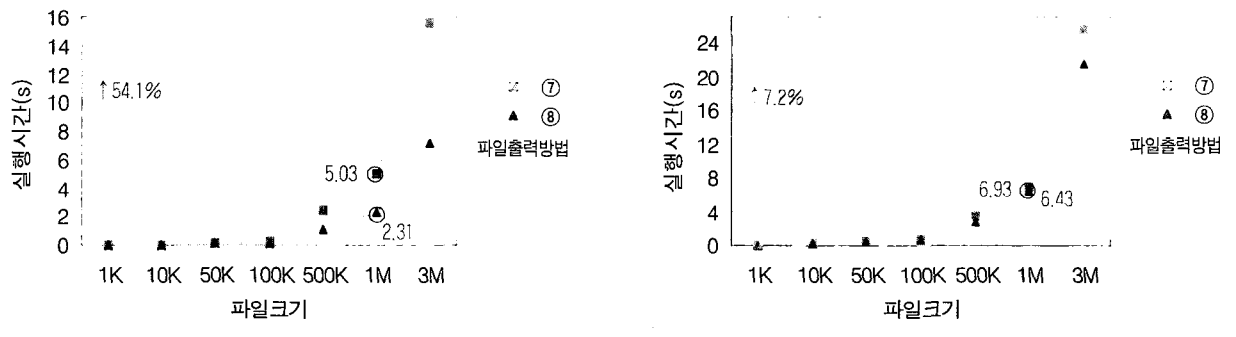
( 그림 11 ) 표준 출력 방법에 따른 운영체제별 실행시간 성능추이

• Digital UNIX((그림 11)의 (d))

버퍼 크기에 따른 영향은 일정하지 않으나 대체로 버퍼의 기본 크기인 8K가 적합하고, 시스템 영역의 버퍼는 도움이 되지 못한다. 또한 기본 출력 라이브러리 대신 (그림 6)의 완전 버퍼 정책의 유형과 관계 없이 방법 ⑤나 ⑥을 사용하면 큰 폭의 성능 향상을 얻을 수 있는데 이는 표준 라이브러리 내에 다양한 부가 기능을 위한 부담이 내재하기 때문이라고 유추된다. 여기서 버퍼 크기 8K와 방법 ⑥으로 얻어지는 이득은 약 10.6%이다.

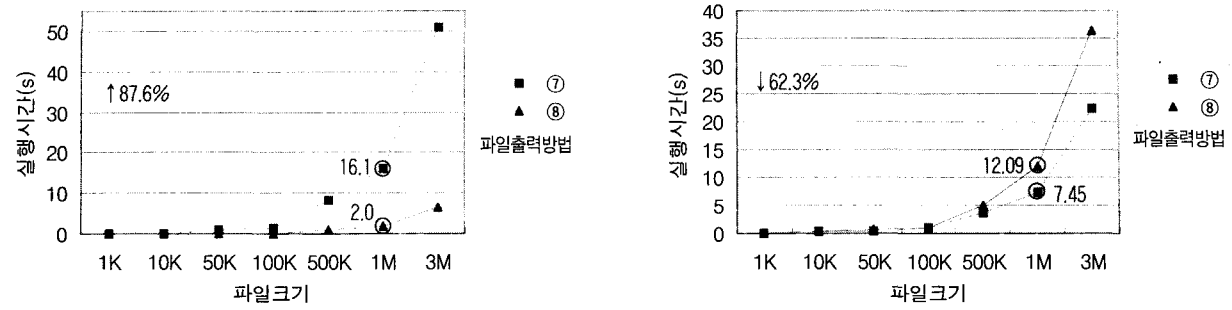
4.2.2 파일 출력 방법에 따른 성능 추이

(그림 12)에 <표 3>에서 분류한 파일 출력 방법 ⑦, ⑧에 따라 파일 크기를 변화시키면서 측정한 각 운영체제별 성능 결과를 보였다. 이 그림으로부터 Digital UNIX를 제외한 모든 운영체제에서 알려진 사실대로 MMF를 이용한 파일 출력 방법 ⑧이 전통적인 방법 ⑦보다 우수하고, 파일 크기가 증가할수록 그 향상 폭은 늘어남을 알 수 있다. 파일 크기가 1M인 경우 Debian LINUX, IBM AIX, SUN Solaris의 성능 이득은 각각 54.1%, 7.2%, 87.6%로서 IBM AIX의 성능이 상대적으로 낮다. 그러나 Digital UNIX에서는 MMF로 인한 성능저하가 62.3%에 이르러 MMF 구현이 매우 특이한 경우로 관찰되었다.




(a) Debian LINUX

(b) IBM AIX




(c) SUN Solaris (d) Digital UNIX

(그림 12) 파일 출력 방법에 따른 운영체제별 실행 시간 성능추이

## 5. 성능 분석 종합 및 실제 환경에서의 검증

### 5.1 성능 분석 종합

(그림 11), (그림 12)의 성능 분석 결과에 따라 <표 3>에 제시된 출력 방법 중 각 운영체제에 가장 적합한 방법을 <표 6>에 요약하였다.

#### 5.1.1 표준 출력 방법의 성능 분석 종합

<표 6>에서 표준 출력 방법에 관한 내용은 아래와 같다.

- **Debian LINUX**

표준 출력 버퍼를 약간 확장하는 것 이외의 다른 개선 사항은 불필요하다.

- **IBM AIX**

표준 출력 버퍼의 크기는 기본 값을 그대로 유지하되, 버퍼를 시스템 영역(MMF)에서 할당한다.

- **SUN Solaris**

표준 출력 버퍼의 크기는 기본 값을 그대로 유지하되, (그림 9)의 (a)를 구현하여 출력 라이브러리로 사용하고 버퍼를 시스템 영역에서 할당한다.

- **Digital UNIX**

표준 출력 버퍼의 크기는 기본 값을 그대로 유지하되, (그림 9)의 (b)를 구현하여 출력 라이브러리로 사용하고 버퍼를 사용자 영역에서 할당한다.

〈표 6〉 운영체제별 최적의 출력 방법

| 운영체제 | 표준 출력 방법 | | 파일 출력 방법 | |
|---|---|---|---|---|
| | 기본방법 | 개선방법 | 기본방법 | 개선방법 |
| Debian LINUX | ①,8K | ①,16K | ⑦ | ⑧ |
| IBM AIX | ①,4K | ②,4K | ⑦ | ⑧ |
| SUN Solaris | ①,1K | ⑤,1K | ⑦ | ⑧ |
| Digital UNIX | ①,8K | ④,8K | ⑦ | x (⑧) |

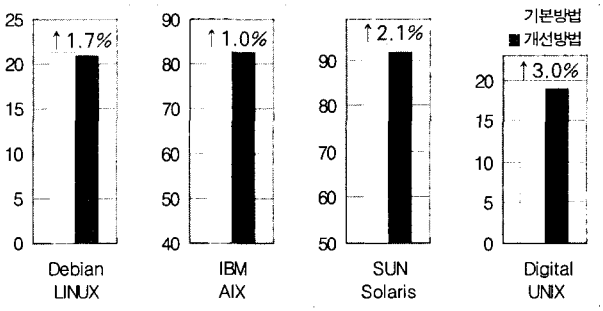
#### 5.1.2 파일 출력 방법의 성능 분석 종합

<표 6>의 파일 출력 방법에서 Digital UNIX를 제외한 다른 모든 운영체제에서 MMF를 사용하는 방법이 우수하다. 또한 그 개선 정도는 출력되는 파일의 크기에 비례한다.

### 5.2 실제 환경에서의 검증

#### 5.2.1 표준 출력 방법의 검증

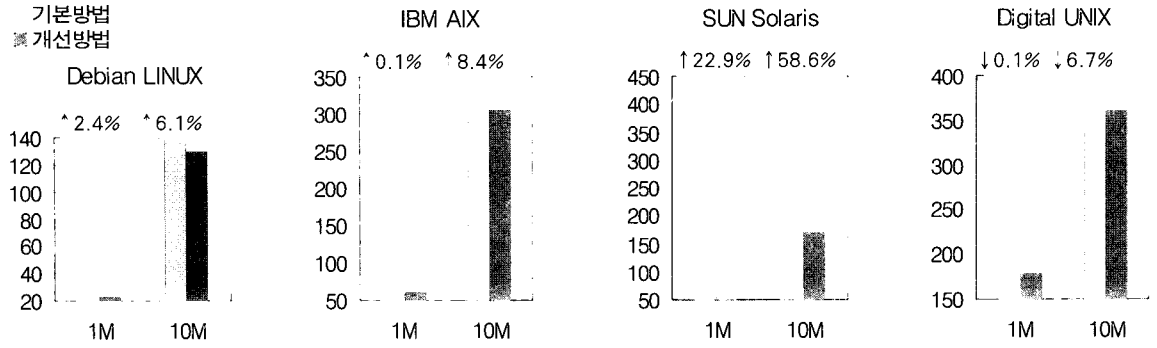
(그림 11)의 성능 측정에 사용된 각 운영체제별 게이트웨이 중에서 <표 6>에 제시된 방법들을 각 시스템에 설치된 실제의 웹 서버를 거쳐 동일한 내용((그림 10)의 게시 글 5페이지)을 내려 받을 때 소요되는 총 시간, 즉 완료 시간을 (그림 13)에 표시하였다. 이 때 웹 클라이언트로는 [18]에 사용된 HTTP 요구 발생기를 사용하였다. 또한 여기서 실행 시간을 측정하지 못하고 완료 시간을 측정한 이유는 웹 서버를 거칠 경우 그 자식 프로세스로 동작하는 게이트웨이의 정확한 실행 시간을 측정할 수 없기 때문이다. (그림 13)은 실제 환경에서도 각각의 개선된 표준 출력 방법이 모든 운영체제에서 효과가 있음을 보이고 있다. 다만 그 향상 정도가 (그림 11)의 실행 시간에서 보다 낮은 이유는 완료 시간에 포함된 많은 대기 시간 때문에 실행 시간에서 얻어진 이득의 비율이 상대적으로 낮아지기 때문이다.




(그림 13) 〈표 6〉의 개선된 표준 출력 방법들의 웹 환경에서의 완료 시간 성능추이

#### 5.2.2 파일 출력 방법의 검증

<표 6>의 파일 출력 방법에서 기본 방법과 개선된 방법으로 각 운영체제별 실제 웹 환경에서 1M, 10M 두 파일을 CGI 게이트웨이로 내려 받는데 소요된 완료 시간을




( 그림 14 ) <표 6>의 개선된 파일 출력 방법들의 웹 환경에서의 완료 시간 성능추이

(그림 14)에 보였다. (그림 12)의 실행 시간 성능에서와 마찬가지로 Digital UNIX를 제외한 모든 운영체제에서 개선된 방법의 효과를 보이고 있다. 또한 표준 출력 방법의 검증에서와 마찬가지로 완료 시간에서의 성능 향상 정도가 실행 시간에서 보다 낮게 보이는데 그 이유 또한 동일하다.

## 6. 결 론

웹 서비스 성능 향상을 위한 가장 쉬운 접근 방안으로 네트워크나 서버 시스템 등 하드웨어를 추가하는 방법을 들 수 있으나 이 경우 구입 및 운영 비용이 적지 않다. 그러나 이 논문에서 제안하고 검증한 출력 방법들에 의한 CGI 게이트웨이 성능 향상은 그 효과의 고저를 떠나 추가 비용이 필요치 않다는 점에서 의의가 있다고 본다. 동일한 출력 방법이라도 유닉스 운영체제의 구현 특성에 따라 그 효과가 각각 달라진다는 사실을 실증적으로 밝히고 운영체제 별로 가장 효과적인 출력 방법을 도출하여 검증했다는 점 또한 매우 의미 있는 일이라 본다.

표준 출력 분야에서 시험 대상이었던 모든 운영체제에서 2~10% 정도의 성능 향상을 이루었고, 파일 출력 분야에서는 어느 운영체제에서도 성능 향상이 기대되었던 MMF에 의한 출력 방법이 오히려 커다란 성능 저하를 초래하는 운영체제가 있다는 사실을 발견하였다. 특히 SUN Solaris의 MMF는 표준 출력 및 파일 출력 분야에서 상당히 높은 성능 이득 기여도를 보인 것으로 나타났다.

이 논문에서 시도한 다양한 출력 방법 및 시험 방법론은 일선 실무 개발 환경에 사용 중인 운영체제에 그대로 적용되어 성능 향상에 기여하거나, 또 다른 출력 방법을 고안하기 위한 토대가 될 수 있을 것으로 믿는다.

## 참 고 문 헌

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

[2] Robert Orfali, Dan Harkey, Jeri Edwards, 'The Essential Client/Server Survival Guide', 2nd Ed., WILEY, p.7~22, 1996.

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

[4] H. Braun and K. Claffy, "Web Traffic Characterization : An Assessment of the Impact of Caching Documents from NCSA's Web Server", Electronic Proceedings of the Second World Wide Web Conference '94 : Mosaic and the Web, Chicago, Ilinois, Oct., 1994.

[5] A. Bestavros, R. Carter, M. Crovella, C. Cunha, A. Heddaya and S. Mirdad, "Application-Level Document Caching in the Internet", Proceedings of the Second International Workshop on Services in Distributed and Networked Environments(SDNE'95), Whistler, BC, Canada, pp.166~173, Jun, 1995.

[6] Arun Iyengar, Eric Nahum, Anees Shaikh, Renu Tewari, "Enhancing Web Performance", IPIP World Computer Congress, Montreal, Canada, Aug., 2002.

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

[8] Arun Iyengar, Jim Challenger, Daniel Dias, Paul Dantzig, "High-Performance Web Site Design Techniques", IEEE Internet Computing, 4(2), Mar./Apr., 2000.

[9] 김성수, 정지영, "웹 서버 클러스터를 위한 효율적인 부하 분배 알고리즘", 정보과학회논문지 : 정보통신, 제 28권 제4호, pp.550~558, 2001.

[10] James C. Hu, Sumedh Mungee, Douglas C. Schmidt, "Techniques for Developing and Measuring High Performance Web servers over High Speed Networks", Proceedings of the 2nd Global Internet Conference, Phoenix, AZ, Nov., 1997.

[11] Yiming Hu, Ashwini Nanda and Qing Yang, "Measurement, Analysis and Performance Improvement of the Apache Web Server", Proceedings of the 18th IEEE

International Performance, Computing and Communication Conference, Phoenix/Scottsdale, Arizona, Feb., 1999.

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

[13] Philippe Joubert, Robert B. King, Rich Neves, Mark Russinovich, John M. Tracey, "High-Performance Memory-Based Web Servers : Kernel and User-space Performance", Proceedings of the USENIX Annual Technical Conference, Boston, MA, Jun, 2001.

[14] Vivek S. Pai, Peter Druschel and Willy Zwaenepoel, "I/O Lite : A copy-free UNIX I/O system", 3rd UNENIX Symposium on Operating Systems Design and Implementation, New Orlenas, LA, Feb., 1999.

[15] Eric Nahum, TsipoSra Barzilai, Dilip Kandlur, "Per-formance Issues in WWW Servers", IEEE/ACM Transactions on Networking, 10(2):2-11, Feb., 2002.

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

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

[18] 이형봉, "유닉스 시스템에서 효율적인 CGI 게이트웨이", 정보과학회논문지 : 컴퓨팅의 실제, 제10권 제1호, pp.55-74, 2004.

[19] W. Richard Stevens, 'Advanced Programming in the UNIX® Environment', pp.132-133, Addison Wesley, 1992.

[20] Digital UNIX, 'Reference Pages Section 3 : Routines Volume 3', p.1-499~1-502, Digital Press, 1996.

[21] Uresh Vahalla, 'UNIX Internals-the new frontiers', pp.437~471, Prentice Hall, 1996.

**이 형 봉**

e-mail : hblee@kangnung.ac.kr

1984년 서울대학교 계산통계학과(이학사)

1986년 서울대학교 대학원 계산통계학(전산과학)과(이학석사)

2002년 강원대학교 대학원 컴퓨터과학과(이학박사)

1986년~1994년 LG전자 컴퓨터 연구소 선임연구원

1994년~1999년 한국디지탈이큅먼트㈜ 책임컨설턴트

1997년~1999년 전자계산조직응용, 전자계산기, 정보통신기술사

1999년~2004년 호남대학교 정보통신공학부 조교수

2004년~현재 강릉대학교 컴퓨터공학과 조교수

관심분야 : 프로그램 언어 및 보안, 운영체제, 알고리즘, 멀티미디어 통신 등

**정 연 철**

e-mail : ycjeong@honam.ac.kr

1991년 전남대학교 전산통계학과(이학사)

1993년 전남대학교 대학원 전산학과(이학석사)

1999년 전남대학교 대학원 전산학과(이학박사)

2001년~현재 호남대학교 컴퓨터게임학과 조교수

관심분야 : 게임 프로그래밍, 게임 기획, 온라인게임 제작, 가상현실, VFX 등

**권 기 현**

e-mail : kweon@samcheok.ac.kr

1993년 강원대학교 전자계산학과(이학사)

1995년 강원대학교 대학원 전자계산학과(이학석사)

2000년 강원대학교 대학원 컴퓨터과학과(이학박사)

1998년~2002년 동원대학 인터넷정보과 교수

2002년~현재 삼척대학교 정보통신공학과 조교수

관심분야 : 분산 시스템, 미들웨어, 임베디드 소프트웨어 등