응답시간 향상을 위한 커넥션 스케줄링 기법
(Connection Scheduling for Improving the Response Time)

방 지 호 † 하 란 ††
(Jiho Bang) (Rhan Ha)

요 약 웹서버가 클라이언트들의 요청을 처리하는 방법은 요청된 문서의 크기를 고려한 스케줄링 기법과 요청된 문서의 크기를 고려하지 않는 스케줄링 기법으로 구분할 수 있다. 웹 문서의 크기를 고려한 스케줄링 기법은 크기를 고려하지 않은 스케줄링 기법 보다 평균 응답시간이 우수하다. 크기기반의 스케줄링 기법인 SRPT(Shortest Remaining Processing Time first)는 웹 문서의 크기를 고려한 대표적인 스케줄링 기법으로 대부분의 연구들이 SRPT 스케줄링 기법을 기반으로 하고 있다. 그러나, 기존 연구들 대부분은 HTTP/1.0 기반으로 클라이언트들이 각각의 커넥션을 통해 HTML 문서에 포함된 다수의 문서들을 동시에 요청할 수 있는 HTTP/1.1 프로토콜의 특성을 고려하지 않았다. 본 논문은 HTTP/1.1 프로토콜을 기반에서 스케줄링 윈도우를 사용하여 커넥션의 전체 응답시간을 향상시키는 커넥션 스케줄링 기법을 제시하고, 성능분석을 통해 제안한 커넥션 스케줄링 기법의 커넥션 응답시간이 SRPT를 이용한 커넥션 스케줄링보다 10% 정도 향상됨을 보인다.

키워드 : 웹서버, 커넥션 스케줄링, 내포된 문서

Abstract The client request scheduling techniques for web server can classified into the scheduling techniques considering a document size to be requested and not. The scheduling techniques considering a document size to be requested provides a better average response time than another. As the size-based SRPT(Shortest Remaining Processing Time first) is typical, and the most of scheduling techniques considering a web document size are based on SRPT. Most of existing researches, however, have not considered the feature of HTTP/1.1 which enable the clients to request concurrent multiple inlined-contents in a HTML document via each connection. In this paper we propose a connection scheduling technique with the scheduling window which provides a better response time in HTTP/1.1. The experimental results show that the performance with the proposed approach is improved about 10% more than the connection scheduling with SRPT.

Key words : Web Server, Connection Scheduling, Connection Response Time

1. 서 론

일반 대중매체인 TV나 신문, 라디오를 통해 얻을 수 있는 정보들을 인터넷을 통해 보다 쉽게 접할 수 있게 되고, 보다 다양한 정보들을 얻을 수 있게 됨에 따라 인터넷 사용 인구가 계속 증가하고 있다. 따라서, 인기가 높은 웹 사이트의 경우 동시에 많은 커넥션(connection, 통신을 목적으로 하는 두 프로그램들 사이에 형성되는 트랜스포트 계층의 가상회로)[1]들이 형성되어 해당 웹 사이트에 접속한 클라이언트들은 서비스 지연을 경험하게 된다. 접속이 빈번한 웹 사이트에 접속하는 클라이언트가 경험할 수 있는 서비스 지연을 효과적으로 개선하기 위해서 해당 웹 사이트의 웹서버는 동시에 형성되는 커넥션들을 빠르게 처리해야 한다. 또는 프록시 서버나 캐시서버를 이용해서 웹서버의 부하를 감소시키면서 클라이언트에 대한 서비스 지연을 효율적으로 감소시킬 수 있다. 그러나, 여기서는 프록시 서버나 캐시서버를 고려하지 않고 웹 서버의 스케줄링 기법에 대해서만 다루도록 한다. 커넥션의 효율적인 처리는 웹서버의 성능에 필수요소이므로 커넥션을 효율적으로 처리하기 위한 다양한 스케줄링 기법들이 연구되고 있다. 웹서버에 요청되는 문서는 정적 문서와 동적 문서로 구분할 수 있다. 정적 문서는 텍스트나 이미지 문서로 클라이언트가 요청하면 웹서버는 서버에 저장된 상태 그대로 클라이언트에게 응답을 한다. 반면 동적 문서는 CGI(Common

· 본 논문은 2004학년도 홍익대학교 교내연구비에 의하여 지원되었습니다.
† 정 회 원 : 홍익대학교 컴퓨터공학과
hallel@empal.com
†† 종신회원 : 홍익대학교 컴퓨터공학과 교수
rhanha@cs.hongik.ac.kr
논문접수 : 2005년 4월 21일
심사완료 : 2005년 9월 28일

Gateway Interface)나 ASP(Active Server Page) 문서로 클라이언트가 요청하면 웹서버는 동적 문서의 실행된 결과를 정적 문서로 변환하여 클라이언트에게 응답을 한다. 따라서 동적 문서의 실행 시간은 항상 변화의 가능성이 있으며, 동적 문서의 크기에 따른 실행 시간을 예측할 수 없기 때문에 동적 문서의 크기를 스케줄링 기법에 이용할 수 없다. 따라서 문서의 크기를 이용하는 스케줄링 기법은 정적 문서에 국한되어 있으며[2] 동적 문서의 경우 각 동적 문서의 실행시간에 대한 로그 정보를 이용하는 별도의 스케줄링 기법이 필요하다.

기존 연구들은 요청된 문서의 크기를 고려한 스케줄링 기법[2-6]과 요청된 문서의 크기를 고려하지 않는 스케줄링 기법[7-11]으로 나누어 볼 수 있다. 전자의 스케줄링 기법은 요청된 문서 중 크기가 작은 문서를 크기가 큰 문서보다 먼저 처리하기 때문에 요청한 문서에 대한 평균 응답시간이 향상되었다. 만약 요청한 문서의 처리시간에 대한 정보가 있다면 클라이언트에서 요청한 문서의 처리시간에 대한 정보를 이용해서 EDF(Earliest Deadline First) 스케줄링 기법으로 스케줄링을 하는 것이 최적이다. 그러나, 서버의 CPU 성능과 메모리, 그리고 처리중인 작업의 양에 따라 문서의 처리시간이 유동적으로 변하기 때문에 문서의 정확한 처리시간을 예측하는 것은 어렵다. 일반적으로 문서의 처리시간은 문서의 크기와 비례하기 때문에 문서의 크기를 이용하면 비슷한 결과를 산출할 수 있다. 문서의 크기를 고려한 스케줄링 기법으로는 SRPT(Shortest Remaining Processing Time first)[2-6]가 대표적이다. 문서의 크기를 고려하지 않는 후자의 스케줄링 기법으로는 운영체제 레벨의 스케줄러에 의존하는 방법과 웹서버 자체의 스케줄러에 의해서 스케줄링 하는 방법이 있다. 아파치(Apache)[9]와 같은 기존의 웹서버는 문서의 크기를 고려하지 않는다. 아파치 웹서버는 멀티 프로세스 방식으로 클라이언트와 형성된 커넥션 당 하나의 프로세스가 할당되어 클라이언트의 요청을 처리한다. 그리고, 각 프로세스들은 운영체제 스케줄러에 의해 스케줄링 된다. 아파치 웹서버와 같은 운영체제 레벨의 스케줄러의 경우 일반 프로세스나 웹서버의 데몬 프로세스에 의해 생성된 프로세스들을 동일 시 하기 때문에 요청된 문서의 특성은 전혀 고려되지 않는 단점이 있다. 따라서 운영체제의 스케줄러에 의존하지 않고 웹서버의 스케줄러가 웹서버의 데몬 프로세스에 의해 생성된 프로세스들을 직접 관리하면서 요청된 문서의 특성에 따라 스케줄링 하는 방법[7-11]이 제안되었다.

HTTP/1.1[1]은 서버나 클라이언트가 TCP 커넥션의 종료 신호를 보내기 전까지 TCP 커넥션을 유지하는 persistent connection과 클라이언트가 서버로부터 각각의 응답을 기다리지 않고 HTML 문서 내에 포함된 여러 문서들의 요청을 보내는 pipelining을 제공한다. 클라이언트가 HTML 문서 내에 포함된 문서들을 요청할 경우, HTTP/1.1 서버는 클라이언트와 연결된 단일 커넥션으로 여러 개의 문서 요청을 받고 응답을 할 수 있으며, 클라이언트는 pipelining을 통해서 HTML 문서 내에 포함된 여러 문서들을 동시에 요청하고 응답 받을 수 있다.

본 논문은 HTTP/1.1에서 다수의 문서가 요청되었을 때 SRPT를 이용한 커넥션 스케줄링 기법으로 개선된 응답시간을 보장할 수 없기 때문에 스케줄링 윈도우의 사용을 통해 개선된 커넥션 응답시간을 제공하는 웹서버의 커넥션 스케줄링 기법을 제시한다.

논문의 구성은 다음과 같다. 2장에서는 관련연구를 설명하고, 3장에서는 HTTP/1.1에서 개선된 응답시간을 나타내는 커넥션 스케줄링 기법을 제안한다. 4장에서는 실험을 통해 제안된 기법의 성능을 평가 분석하고, 5장에서 결론을 맺는다.

2. 관련연구

2.1 요청된 문서의 크기를 고려한 스케줄링 기법

M.E.Crovella et al.[2]는 SRPT를 이용한 커넥션 스케줄링(이하, SRPT-CS)을 제안하였다. SRPT-CS는 클라이언트와 커넥션이 형성된 후에 커넥션이 서비스 받는 과정을 프로토콜 큐, 디스크 큐, 그리고 네트워크 큐를 통해서 구성하였다. 클라이언트들의 요청은 처음에 프로토콜 큐에 할당된다. 그리고, 프로토콜 큐에 있는 요청들은 어떤 문서에 대한 요청인지 모르기 때문에 들어온 순서(FIFO)로 서비스 받아서 큐에 있는 요청들은 요청한 문서의 크기 정보를 얻는다. 이후, 요청한 문서의 크기에 따라 디스크 큐에 저장된다. 요청된 문서의 크기와 현재 처리가 남은 데이타양에 따라 네트워크 큐에 저장되어 스케줄링 된다. 따라서 문서의 크기가 작거나 처리가 남은 데이타의 양이 적을수록 빨리 스케줄링 되기 때문에 문서에 대한 평균 응답시간이 향상되었으며, 문서의 크기를 고려하지 않는 스케줄링 기법에 비해 평균 응답시간이 향상되었다. 동일 커넥션으로 상이한 크기의 여러 문서가 동시에 요청되었을 경우, SRPT-CS의 스케줄링의 범위가 전체 커넥션으로 확장되어 각 커넥션을 처리하므로 커넥션별 전체 응답시간의 향상을 보장할 수 없다. M.Harchol-Balter et al.[3]은 응용레벨에서 구현되었던 SRPT 알고리즘을 Linux 커널에 구현하였으며, LAN과 WAN 환경에 대해서 실험을 하였다. E.J.Friedman et al.[4]는 SRPT가 PS(Process Sharing)보다 평균 응답시간이 우수하지만 공정하지 못하며, 상대적으로 크기가 큰 문서에 대한 처리시간이 계

속 지연되어 기아현상을 유발할 수 있는 SRPT의 단점을 개선하고자 FSP(Fair Sojurn Protocol)[4-6]를 제안하였다. FSP는 PS에 따른 처리시간을 기준으로 처리시간이 짧은 요청을 먼저 처리해서 SRPT보다 평균 처리시간은 늦지만 SRPT가 경험할 수 있는 기아현상을 막을 수 있다. D.Lu et al.[5,6]는 크기기반의 스케줄링 알고리즘인 FSP와 SRPT에 대해서 문서의 크기와 처리시간의 관계에 초점을 두고 실험을 하였다. 과거의 요청을 이용한 SRPT-D(SRPT Domain-based)과 FSP-D (FSP Domain-based)를 제안[5]하였다.

2.2 요청된 문서의 크기를 고려하지 않는 스케줄링 기법

N.Bhatti et al.[7]에서 제안한 웹서버는 커넥션 매니저, 요청 분류자, 허가 제어기, 그리고 요청 스케줄러로 구성된다. 커넥션 매니저는 웹서버의 로드를 고려해서 우선순위가 높은 요청의 서비스를 보장하는 허가 제어기와 클라이언트의 주소나 HTTP 쿠키 등에 따라 요청된 문서를 분류하는 요청 분류자의 정보를 이용해서 클라이언트의 요청을 받아들이고, 서로 다른 우선순위가 부여된 큐로 요청을 할당한다. 그리고, 요청 스케줄러는 우선순위에 따라 서로 다른 큐에 있는 클라이언트들의 요청을 스케줄링 한다. J.Almeida et al.[8]는 요청된 문서를 기반으로 할당된 우선순위에 따라 사용자 레벨에서 non-work conserving 방법, 또는 커널 레벨에서 work conserving 방법으로 스케줄링을 하는 기법을 제안하였다. 그리고, P.Bhoj et al.[9]의 웹서버의 수용자는 N.Bhatti et al.[7]의 커넥션 매니저처럼 우선순위가 다른 두 개의 큐로 클라이언트의 요청을 할당하여 우선순위에 따라 요청이 처리되게 한다. M.Bender et al.[10]에서는 클라이언트들의 각 요청마다 마감 시한을 부여해서 마감 시한이 빠른 요청부터 스케줄링을 한다. 그리고, 서비스 지연에 대한 피드백 정보를 가지고 커넥션을 스케줄링 하는 기법[11]이 있다. 우선순위에 따른 스케줄링 기법은 전자상거래 사이트 등의 웹서버에 적합하지만, 우선순위가 낮은 요청들은 상대적으로 우선순위가 높은 요청들 때문에 계속 처리가 지연되고, 서버의 메모리에 남게 된다. 따라서 수용할 수 있는 커넥션의 수가 감소하게 되고, 우선순위가 낮은 요청들의 지연으로 전체 처리시간이 길어지게 된다.

3. 커넥션 스케줄링 기법

HTTP/1.1에서 클라이언트들은 pipelining을 통해 HTML 문서 내에 포함된 문서들을 동시에 요청할 수 있다. 이 장에서는 본 논문에서 제안하는 윈도우 기반의 커넥션 스케줄링(Window-based Connection Scheduling, 이하 WCS) 기법에 대해서 살펴본다. 먼저 WCS 알고리즘의 구현을 위한 웹서버 모델과 WCS 알고리즘에서 사용되는 스케줄링 윈도우에 대해 설명한다. 그리고, WCS 알고리즘과 WCS 알고리즘이 적용된 예를 설명한다.

3.1 스케줄링 윈도우

그림 1은 WCS 알고리즘이 구현된 웹서버 모델을 나타낸다. 그림 1에서 커넥션 스케줄러는 pipelining을 통해 동시에 여러 개의 문서를 요청하는 클라이언트들과 커넥션이 형성되면 몇 개의 문서를 요청했으며, 어떤 문서를 요청했는가에 대한 커넥션 정보를 점검하고 저장한다. 그리고, 클라이언트가 동시에 요청한 문서들의 전체 크기에 의해 커넥션 정보를 스케줄링 되는 순서로 정렬한다. 커넥션 스케줄러는 동시에 요청된 문서들의 전체 크기가 가장 작은 커넥션 정보에 따라 스케줄링의 범위가 되는 스케줄링 윈도우의 크기(Scheduling Window Size, 이하 Δ)를 설정한다. 그리고 스케줄링 윈도우의 범위 안에 있는 각 커넥션을 처리한다.

본 논문에서 제안하는 WCS는 스케줄링 윈도우를 사용함으로써 커넥션의 응답 시간을 향상시키고, 처리를 기다리는 커넥션들에게 제한적인 공평성을 제공할 수 있다. 스케줄링 윈도우는 처음 스케줄링이 시작되는 커넥션으로 요청된 문서 수에 따라 Δ를 설정하는 방법(이하 Δ_P) 또는 큐에 쌓인 전체 요청 수에 비례해서 전체

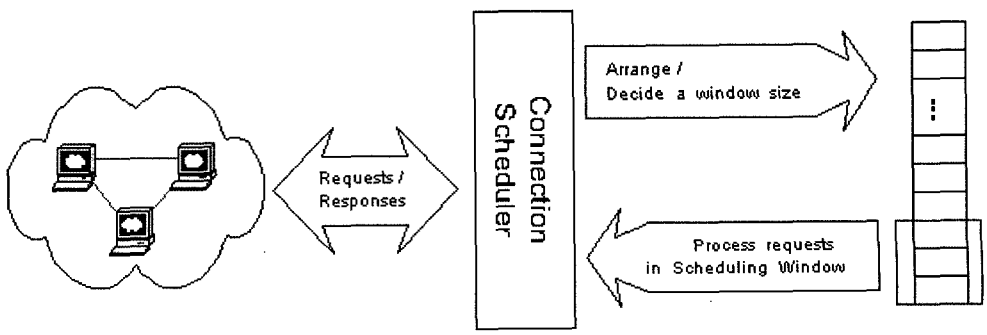


그림 1 웹 서버 모델

요청의 평균으로 Δ를 설정하는 방법(이하 Δ_Q)으로 설정될 수 있다. 4장에서 Δ_P와 Δ_Q에 대한 성능비교로 볼 때, Δ를 현재 처리대상의 커넥션의 요청 수에 따라 스케줄링 윈도우를 설정하는 방법이 전체 커넥션 큐를 조사하여 전체 큐의 평균으로 하는 방법보다 효율적이고 응답시간이 우수하다. 따라서 본 논문에서 제안하는 WCS의 스케줄링 윈도우는 처음 스케줄링이 시작되는 커넥션으로 요청된 문서들의 수에 따라 Δ를 설정하는 방법을 적용하였다.

3.2 WCS 알고리즘

스케줄링 윈도우는 스케줄러가 가장 먼저 스케줄링을 시작하는 커넥션으로 동시에 요청된 문서의 수로 결정한다. 스케줄링 윈도우는 스케줄링의 범위를 나타내기 때문에 Δ만큼 스케줄링 되는 커넥션 수가 결정된다. 만약, 3개의 문서가 동시에 요청되었다면 Δ는 3으로 설정되고 동시에 요청된 문서의 전체 크기가 가장 작은 3개의 커넥션이 스케줄러에 의해 스케줄링 된다. 이때, 커넥션의 크기에 따라 정렬되어 있기 때문에 큐의 처음부터 3번째 위치한 커넥션까지 스케줄링의 대상이 된다. 스케줄링 윈도우에 의해 정해진 범위 내에서 해당되는 각 커넥션을 스케줄러가 라운드 로빈(Round-Robin, 이하 RR) 방식으로 스케줄링 한다. 그리고, 스케줄링의 범위가 정해져서 스케줄링이 진행되고 있는 가운데 스케줄링 범위 밖의 다른 커넥션에 의해 선점되지 않는다. 스케줄러에 의해 큐의 가장 처음에 있는 커넥션의 처리가 끝나면 스케줄러는 다음에 위치한 커넥션의 문서의 전체 요청 수 또는 현재 처리가 남은 문서의 요청 수에 의해 스케줄링 윈도우를 설정하고 스케줄링을 시작한다. 만약, 큐의 처음에 있는 커넥션의 처리가 다 끝나기 전에 스케줄링 윈도우 내의 다른 커넥션이 먼저 처리가 끝나서 해당 커넥션이 종료되면 스케줄러는 스케줄링 윈도우 밖의 다음 커넥션을 스케줄링 대상에 포함시키고 스케줄링을 한다. 표 1은 자세한 WCS 알고리즘이다.

표 2는 여러 개의 문서들을 포함하고 있는 *a.html, b.html, c.html, d.html*, 그리고 *e.html*의 5개 HTML 문서들로 구성된 웹사이트를 나타낸다. 표 2에서 *a.html* 문서 내에 포함된 문서들을 *a.class, b.html* 문서 내에 포함된 문서들을 *b.class* 문서 내에 포함된 문서들을 *b.class, c.html* 문서 내에 포함된 문서들 *c.class, d.html* 문서 내에 포함된 문서들을 *d.class*, 그리고 *e.html* 문서 내에 포함된 문서들을 *e.class*라고 하자. 그리고, 각 문서들을 요청하는 10의 클라이언트들을 *A, B, C, D, E, F, G, H, I*라 하자. *A, C, E, G, I* 클라이언트들은 GET method로 각각 *a.html, b.html, c.html, d.html, e.html* 문서들을 요청했고, 이전에 각각의 HTML 문서를 요청했던 *B, D, F, H, J* 클라이언트들은 각각의

표 1 WCS 알고리즘

```
PROCEDURE WCS (first)
  Δ = c_des[first].total_req;
  // first : 가장 크기가 작은 커넥션, 큐의 맨 앞에 위치
  // Δ는 처음 스케줄링이 시작되는 커넥션으로 요청된 정적
     문서들의 수
  while (c_des[first]->file is not NULL)
    do
      for (i=0;i<Δ;i++)
        read (c_des[i]->file);
        write (c_des[i]->file);
    while (c_des[0]->file || ... || c_des[Δ-1]->file is not EOF)
    for (i=0;i<Δ;i++) //i=0,1,...,Δ-1
      if (c\_des[i] is not NULL)
        c_des[i]->file = c_des[i]->file->next;
        c_des[i].total_req--;
      else
        c_des[i-1]->next = c_des[i]->next;
        remove (c_des[i]);
        add_schedule (c_des[Δ]);
END PROCEDURE
```

표 2 각 정적 문서들의 크기

HTML 문서(bytes)	HTML 문서 내에 포함된 정적 문서(bytes)
a.html(3177)	*a_1.gif(218), a_2.gif(2234), a_3.gif(2046), a_4.gif(3958)*
b.html(102)	*b_1.gif(2223), b_2.gif(13786)*
c.html(1774)	*c_1.gif(1956), c_2.gif(13786)*
d.html(2153)	*d_1.jpg(15671), d_2.gif(13786)*
e.html(3638)	*e_1.jpg(1356), e_2.gif(10703), e_3.jpg(135075)*

HTML 문서 내에 포함된 *a.class, b.class, c.class, d.class, e.class*의 문서들을 요청했다. 이때, 각각의 HTML 문서 내에 포함된 각 class의 크기는 62236, 16009, 15742, 29457, 147134 바이트이다. 커넥션의 크기에 의해서 정렬된 순서는 *C, E, G, A, I, F, D, H, B, J*이다. 큐의 앞에서부터 *C, E, G, A, I* 클라이언트에 의해서 한 개의 HTML 문서가 요청되었기 때문에 WCS 알고리즘에 의해서 Δ가 1로 설정되고, *C, E, G, A, I* 클라이언트 순서로 스케줄링 된다. 그림 2와 같이 C 클라이언트의 *b.html* 문서가 처리되고, 계속 I 클라이언트의 *e.html* 문서가 처리될 때까지 Δ는 1인 상태로 해서 각각의 HTML 문서가 처리된다. 그 다음에 위치한 *F, D, H, B, J* 클라이언트는 요청한 문서의 수가 2 이상이기 때문에 계속해서 Δ가 변하게 된다. 큐의 앞쪽에 위치한 F 클라이언트에서 2개의 문서(*c1.gif, c2.gif*)가 요청되었기 때문에 WCS 알고리즘에 의해 Δ는 2로 설정되고 스케줄링의 범위는 *F*부터 *D* 클라이언트까지가 된다. 그림 2에서 RR 스케줄링 기법으로 *F*와 *D* 클

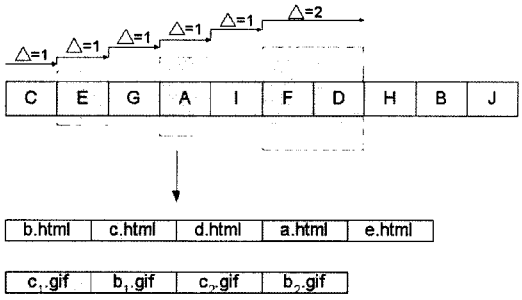


그림 2 HTML 문서의 처리(Δ=1, 2)

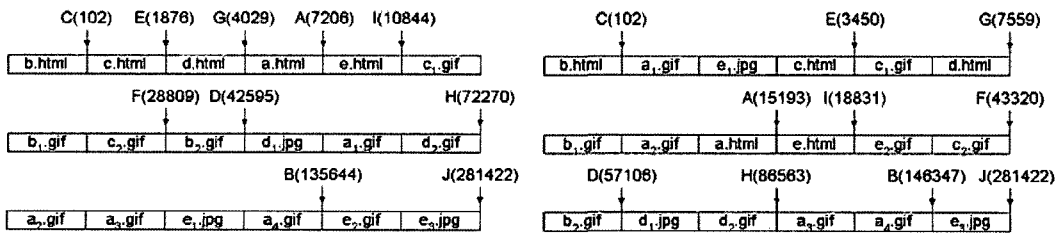
라이언트에서 요청된 문서들이 처리된 것을 보여준다. *F* 클라이언트에서 요청된 문서들이 모두 처리됨과 동시에 *D* 클라이언트에서 요청된 문서도 다 처리되기 때문에 다음에 위치한 *H* 클라이언트로 스케줄링 시점이 넘어가게 된다. *H* 클라이언트에서 2개의 문서가 요청되었기 때문에 Δ는 2로 설정되고, *B* 클라이언트까지 스케줄링의 범위가 된다. WCS 알고리즘에 의해서 *H* 클라이언트에서 요청된 문서가 모두 처리되었을 때 *B* 클라이언트에서 요청된 문서는 4개 중 2개의 문서(*a1.gif, a2.gif*)가 처리되고 2개의 문서(*a3.gif, a4.gif*)가 남게 된다. 그래서, 스케줄링 윈도우는 *H* 클라이언트에서 요청된 문서 중 처리가 남은 2개의 문서 수만큼 설정(Δ=2)된다. *B* 클라이언트부터 *J* 클라이언트까지 스케줄링의 범위가 되고 RR 스케줄링 기법에 의해서 처리된다. 그림 3(a)는 WCS 알고리즘으로 스케줄링된 문서들의 순서이다. 그림에 표시된 화살표는 클라이언트에서 요청한 문서들을 모두 처리한 시점을 나타내며, 화살표 옆에 있는 알파벳 기호는 해당 클라이언트를 나타낸다. 그리고, 그림 3(b)는 SRPT-CS 알고리즘에 의해 스케줄링된 문서들의 순서이다. SRPT-CS 알고리즘은 WCS 알고리즘처럼 문서의 크기를 기준으로 스케줄링을 하지만 WCS 알고리즘처럼 커넥션으로 동시에 요청된 문서들의 전체 크기를 기준으로 하지 않고 커넥션에 관계없이 단일 문서의 크기를 기준으로 한다. 그리고, SRPT-CS 알고리즘은 WCS 알고리즘처럼 스케줄링 윈도우를 사용하지 않는다. 따라서 *A, B, C, D, E, F, G, H, I* 클라이언트에 의해 요청된 문서들 중 가장 크기가 작은 문서를 처리해 준다. 그러나, 요청된 순서대로 처리를 해야하기 때문에 하나의 커넥션에서 크기가 큰 문서를 먼저 요청하고 크기가 작은 문서를 그 다음에 요청했을 때, 크기가 큰 문서를 다른 커넥션들의 문서와 크기를 비교한 후 처리를 한다. 그 다음에 요청된 문서는 먼저 요청된 문서가 처리되기 전까지 처리가 지연된다. WCS 알고리즘을 설명하기 위해 사용된 예를 동일하게 적용했기 때문에 그림 3(a)와 3(b)를 통해 WCS 알고리즘과 SRPT-CS 알고리즘의 성능을 직접 보고 비교할 수 있다. 각 클라이언트가 원하는 문서를 서비스 받는데 소요되는 평균 바이트를 보면 WCS 알고리즘은 약 61KB이고, SRPT-CS 알고리즘은 약 66KB이다. 그리고, 그림 3의 화살표와 알파벳 기호는 각 클라이언트가 서비스를 모두 받은 후 종료되는 시점이기 때문에 그림 3을 통해서 각 클라이언트의 종료 시점을 비교할 수 있다. *C* 클라이언트를 제외한 모든 클라이언트의 종료시점은 SRPT-CS 알고리즘에 의한 결과보다 WCS 알고리즘에 의한 결과가 더 빠르게 나타났다. 따라서, 각 클라이언트의 종료시점과 서비스 받은 평균 바이트를 통해 WCS 알고리즘이 SRPT-CS 알고리즘에 비해 우수한 커넥션 응답 시간이 나타날 것이라고 생각할 수 있다. 자세한 성능 평가는 4장에서 설명한다.

4. 성능분석

이 장에서는 WCS 기법의 성능을 평가하기 위한 실험 환경과 방법을 설명한다. 그리고, 제안한 WCS 기법이 다른 스케줄링 기법과 성능 비교된 결과를 설명한다.

4.1 실험환경

성능분석을 위한 네트워크 환경은 10/100MB 대역폭을 갖는 네트워크 상에 본 논문에서 제안한 스케줄링 기법과 비교대상 알고리즘이 구현된 리눅스 서버(Pentium IV, 256MB)와 HTTP/1.1 환경에서 웹 문서를 요청하는 클라이언트(Pentium IV, 256MB)로 구성된다. 다음은 제안한 스케줄링 기법의 구현 및 HTTP/1.1 환



(a) WCS (b) SRPT-CS

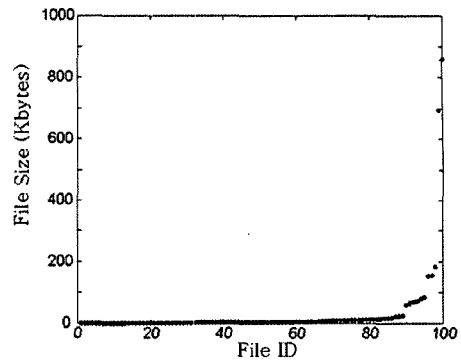
그림 3 WCS, SRPT-CS에 의한 문서 처리 순서

경에서 웹 문서를 요청하는 방법에 대한 자세한 내용이다.

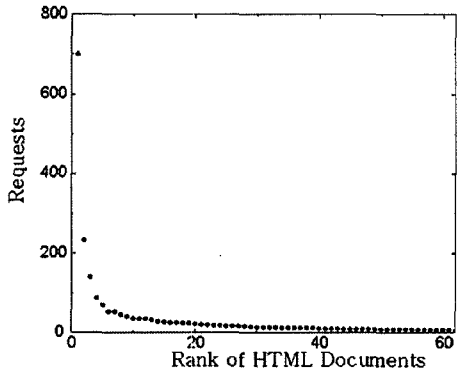
웹서버의 동작방식은 크게 3종류로 설명할 수 있다[14]. 첫 번째는 웹 문서의 요청이 있을 때마다 데몬 프로세스가 프로세스를 생성하고, 생성된 프로세스가 웹 문서의 요청을 처리하는 멀티 프로세스 방식이 있다. 아파치(Apache)[12]는 멀티프로세스의 대표적인 웹서버이다. 두 번째는 데몬 프로세스가 모든 웹 문서의 요청을 처리하는 단일 프로세스 방식이 있다. 마지막으로 단일 프로세스 방식으로 동작을 하면서 디스크 I/O 발생 시 헬퍼(Helper) 프로세스가 디스크 I/O를 담당하는 AMPED (Asymmetric Multi-process Event Driven)방식이 있으며 플래시(Flash)[14]가 대표적인 웹서버이다. 두 번째의 단일 프로세스 방식은 나머지 두 방식에 비해서 스케줄링 기법을 적용하기가 쉽다. 따라서, 본 논문에서 제안한 스케줄링 기법의 성능평가를 위해 플래시 웹서버를 수정하여 구현하였다. 그리고, pipelining 효과를 나타내기 위해 클라이언트들은 단일 문서를 요청하는 GET METHOD 대신 여러 개의 문서를 요청할 수 있는 GETALL METHOD[13]를 사용한다. 클라이언트가 GETALL METHOD를 사용해서 HTML문서를 요청하면 서버는 HTML 문서와 HTML 문서에 포함된 문서들을 모두 서비스해야 한다. 따라서, 클라이언트가 pipelining으로 여러 문서를 동시에 요청한 것처럼 되기 때문에 서버는 단일 TCP 커넥션으로 다수의 응답을 클라이언트에게 보내게 된다.

본 논문에서 제안된 스케줄링 기법인 WCS의 성능평가를 위해 SRPT-CS와 문서의 크기를 고려하지 않는 FIFO를 비교 실험하였다. SRPT-CS는 WCS처럼 문서의 크기를 기준으로 스케줄링을 하지만 단일 문서의 크기를 기준으로 스케줄링을 하기 때문에 스케줄링 윈도우를 사용하고 커넥션으로 동시에 요청된 전체 문서의 크기를 기준으로 스케줄링하는 WCS와는 차이가 있다. 그리고, FIFO는 WCS와 SRPT-CS처럼 문서의 크기를 전혀 고려하지 않고 들어온 순서대로 처리를 하며 각 프로세스의 활성화는 Unix와 같은 운영 체제의 영향을 받기 때문에 WCS와 차이가 있다.

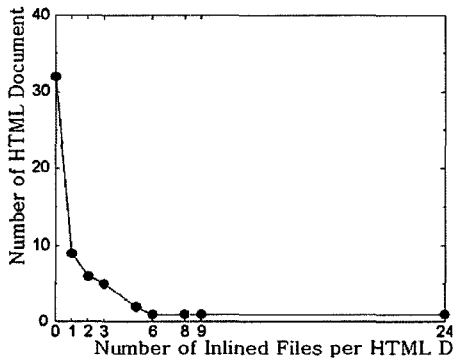
웹서버의 성능을 측정하는데 사용되는 Surge[15]에 의해 Pareto 분포[15,16]를 따르는 문서들과 Zipf-like law[15,16]를 따르는 문서들에 대한 요청들을 생성시키고 커넥션의 수를 500개에서 500개씩 증가시키면서 커넥션에 대한 응답시간을 측정하였다. 그림 4는 Surge가 문서들의 크기 분포와 문서들의 요청 분포를 생성하기 위해 사용된 함수와 Surge에 의해 생성된 100개 문서들의 특성을 나타낸다. 실제 웹 상에 존재하는 문서들은 상대적으로 크기가 작은 문서들이 많이 분포하는데 Pareto 분포와 비슷하며, 문서들의 요청은 상대적으로



(a) 문서의 분포



(b) HTML 문서의 인기 순위



(c) Inlined 문서의 분포

그림 4 Surge를 통해 생성된 문서들의 분포

크기가 작은 문서에 집중되는데 Zipf-like law와 같은 분포를 보인다. 그림 4(a)는 Surge에 의해 생성된 문서들의 크기 분포로 HTML 문서와 HTML 문서에 포함된 문서들로 구성되며, 상대적으로 크기가 작은 문서들이 많은 Pareto 분포를 따르는 것을 볼 수 있다. 그림 4(b)는 Zipf-like law와 같은 분포를 보이는 문서들의 요청 분포를 보여준다. 그림 4(b)에서 인기가 많은 HTML

문서는 최고 700번 요청되며 인기가 적은 HTML 문서는 최저 7번의 요청이 되는 것을 보여준다. 그리고, 그림 4(c)는 Pareto 분포를 따르는 HTML 문서에 포함된 문서들의 분포로 HTML 문서에 포함된 문서의 수가 적은 HTML 문서들이 상대적으로 많은 것을 보여주며, 아무런 문서도 포함하지 않는 HTML 문서는 32개이며, 최고 24개의 문서를 포함하는 HTML 문서는 1개인 것을 보여준다.

4.2 결과분석

그림 5, 6, 7은 웹서버의 성능을 측정하기 위해 개발된 Surge를 이용하여 웹 문서들을 생성하고, 생성된 문서들을 요청하는 클라이언트의 수를 대량으로 증가시키면서 측정한 결과이다. 그림 5는 스케줄링 윈도우를 설정하는 방법에 대한 내용이며, 그림 6과 7은 문서의 크기에 따른 평균 응답시간과 커넥션에 대한 평균 응답시간에 대한 내용이다. 여기서 응답시간은 서버가 pipelining으로 요청받은 시점에서부터 pipelining으로 요청받은 문서들을 모두 처리한 시점까지 걸린 시간이다. 다음은 이에 대한 결과분석이다.

그림 5는 스케줄링 윈도우가 Δ_P와 Δ_Q에 따라 각각 설정되어 성능 비교된 결과를 보여준다. 스케줄링 윈도우가 Δ_P의 방법으로 설정되어 각 커넥션들을 처리했을 때 전체 커넥션의 평균 응답시간이 Δ_Q의 방법으로 설정되어 각 커넥션들을 처리했을 때보다 빠른 응답시간을 보여준다. Δ_Q로 스케줄링 될 때 처음의 Δ의 값은 전체 요청 수에 따라 1에서 2사이의 값이 설정되지만 정렬된 커넥션 큐에서 작은 커넥션들이 처리되고 크기가 큰 커넥션들이 남아 있기 때문에 점차 Δ가 점차 증가하게 되어 스케줄링의 범위가 확장되어진다. 반면, Δ_P는 한 커넥션으로 요청된 문서들의 수에 따라 Δ가 정해지기 때문에 스케줄링의 범위가 크게 변하지 않기 때문에 Δ_P가 Δ_Q에 비해 빠른 응답시간을 보여준다. 그리고, Δ를 현재

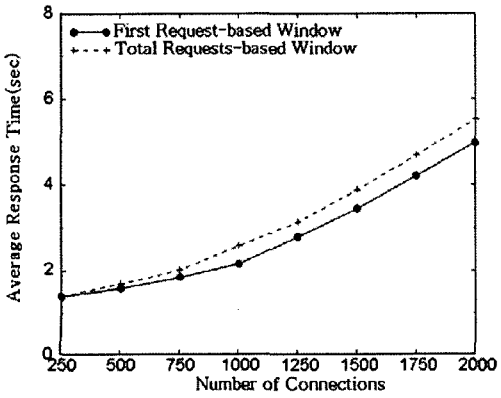
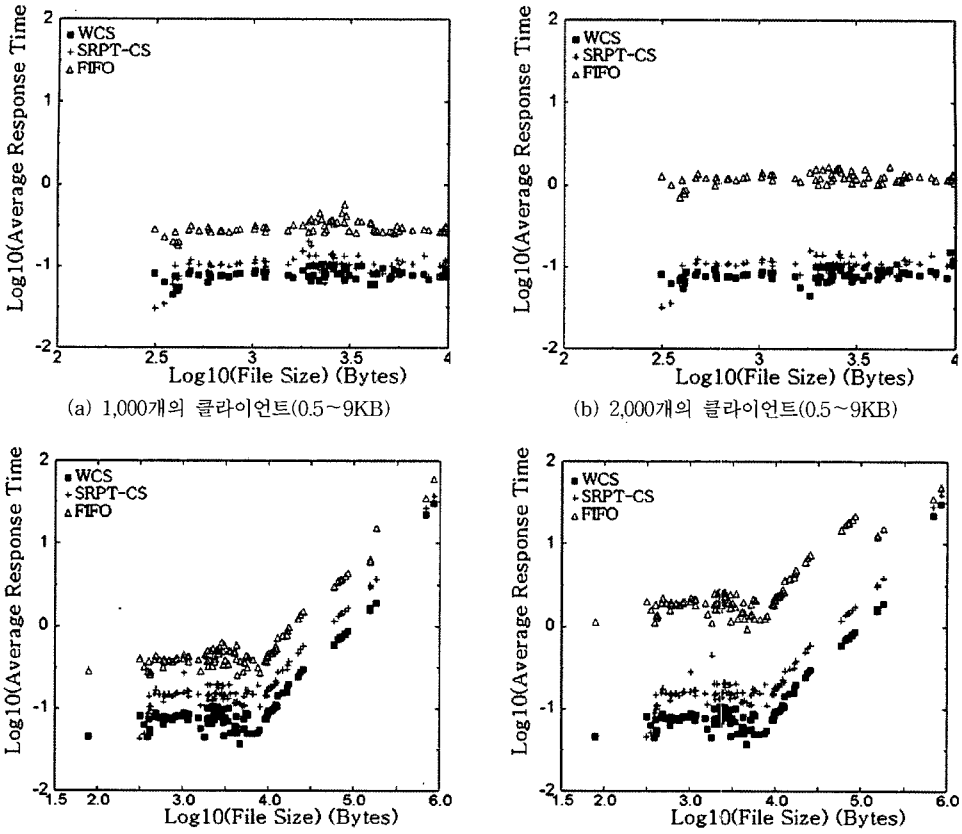


그림 5 스케줄링 윈도우를 설정하는 두 방법의 비교

처리대상의 커넥션의 요청 수에 따라 스케줄링 윈도우를 설정하는 방법이 전체 커넥션 큐를 조사하여 전체 큐의 평균으로 하는 방법 등 다른 방법보다 효율적이고 실험에 의해 그 성능이 우수하다.

그림 6은 클라이언트의 수를 증가시키면서 그림 4의 분포를 따르는 문서들을 요청하고 응답 받은 결과들을 보여준다. 그림 6(a)와 6(b)는 그림 4의 분포를 따르는 문서들 중에서 크기가 *0.5-9KB*인 문서들을 요청했을 때 클라이언트의 수가 1000개 일 때와 2000개 일 때의 문서의 크기에 따른 평균 응답시간을 나타내며, 그림 6(c)와 6(d)는 그림 4의 분포를 따르는 문서들을 클라이언트의 수가 1000개 일 때와 2000개 일 때의 문서의 크기에 따른 평균 응답시간을 나타낸다. 그림 6은 그래프 표현상 혼동이 없도록 하기 위해 크기와 평균 응답시간을 log로 나타내었다. 그림 6(a)와 6(b)는 문서의 크기 차이가 크지 않은 경우에 대한 각 문서의 응답 시간으로 WCS가 FIFO에 비해 빠른 응답시간을 나타냈지만 SRPT-CS와는 비슷한 추이를 보여주었다.

이것은 WCS와 SPRT-CS가 똑같이 문서의 크기를 고려하기 때문에 문서의 크기 차이가 적은 경우에 비슷한 추이를 보인다. 그러나 WCS가 스케줄링 윈도우를 사용하기 때문에 스케줄링의 범위가 제한되는 반면 SRPT-CS는 스케줄링의 범위가 요청된 전체 문서가 되기 때문에 WCS가 좀 더 빠른 응답시간을 나타낸다. 서버의 부하가 높았을 때 요청된 문서의 경우 그림 6(a)의 상대적으로 문서의 크기가 큰 부분에서 다른 문서들과는 달리 응답 시간이 느려는 경우를 볼 수 있다. 그리고, 그림 6(c)와 6(d)는 문서의 크기가 차가 큰 경우에 대한 각 문서의 응답 시간으로 WCS가 SRPT-CS와 FIFO에 비해 빠른 응답 시간을 보여주고 있다. 전체적으로 그림 6에서 문서의 크기에 따른 평균 응답 시간은 제안한 WCS가 SRPT-CS와 FIFO에 비해 빠른 응답 시간을 보이며, 비슷한 응답 시간의 추이를 나타냈다. 그리고, WCS와 SRPT-CS는 클라이언트의 수가 증가하여도 크게 변화가 없었으나, FIFO는 클라이언트의 수가 증가함에 따라 문서의 크기에 대한 평균 응답시간이 더 느려졌으며 변화의 폭도 커졌다. 이것은 SRPT-CS가 크기가 작은 문서를 먼저 처리하기 때문에 커넥션 단위로 처리하는 WCS에 비해 크기가 작은 문서의 경우 처리시간이 빠르다. 그러나 문서의 크기가 커질수록 스케줄링 윈도우를 사용하는 WCS가 일정한 스케줄링을 보장하기 때문에 SRPT-CS에 비해 빠른 응답시간이 나타난다. 그리고, WCS와 SRPT-CS가 크기가 작은 커넥션 또는 크기 작은 문서들을 먼저 처리하지만 FIFO는 크기를 고려하지 않고 요청된 모든 문서들을 RR 기법에 의해 처리하기 때문에 이와 같은 결과가 나



(a) 1,000개의 클라이언트(0.5~9KB)

(b) 2,000개의 클라이언트(0.5~9KB)

(c) 1,000개의 클라이언트(0.07~858KB)

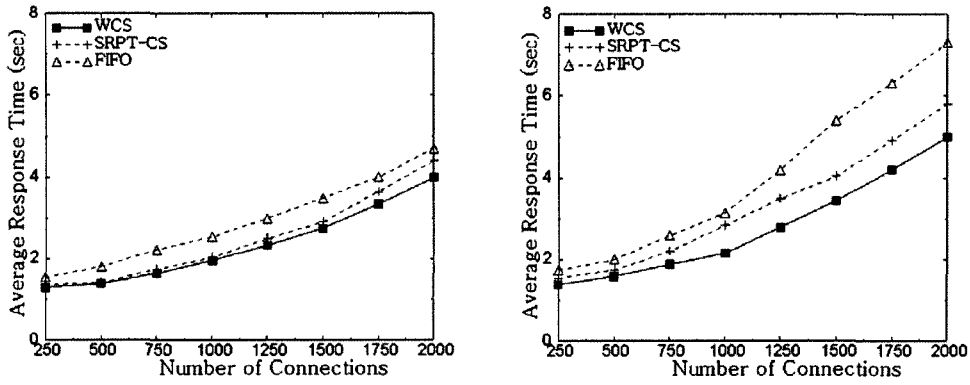
(d) 2,000개의 클라이언트(0.07~858KB)

그림 6 문서의 크기에 따른 평균 응답시간

타났다. 그림 6(c)와 6(d)의 문서 크기가 큰 경우에서는 세 스케줄링 기법의 응답시간이 비슷하게 나타났는데, 이것은 문서의 크기가 큰 경우 각 커넥션의 크기도 커지기 때문에 WCS에서 스케줄링이 지연되며, SRPT-CS는 크기가 작은 문서를 먼저 처리하기 때문에 크기가 큰 문서들은 스케줄링이 지연되고, FIFO는 스케줄링의 범위가 웹 문서를 처리하는 모든 프로세스가 되기 때문에 크기가 큰 문서는 다른 문서와 동등한 조건으로 스케줄링 되지만 크기가 커서 처리되는데 시간이 많이 걸리기 때문이다.

그림 7은 클라이언트의 수가 500개씩 증가할 때마다 평균 커넥션 응답시간으로 그림 7(a)는 그림 5의 분포를 따르는 문서들 중 크기가 *0.5-9KB*인 문서들을 요청했을 때의 결과이고, 그림 7(b)는 그림 4의 분포를 따르는 문서들을 크기가 *0.07-858KB*인 문서들을 요청했을 때의 결과이다. 그림 7(a)는 그림 7(b)에 비해 문서의 크기 차이가 작아서 문서의 크기를 고려한 스케줄링 기법이나 문서의 크기를 고려하지 않는 스케줄링 기법의 차이가 크지 않기 때문에 전체 커넥션에 대한 평균 응답 시간의 변화가 완만하게 나타났다. 그러나 그림 7(b)는 문서의 크기 차이가 크기 때문에 크기가 문서의 크기를 고려하는 스케줄링 기법을 적용하여 크기가 작은 문서를 먼저 처리함으로써 응답시간의 개선을 가져올 수 있다. 그림 7(a)와 7(b)에서 WCS가 SRPT-CS와 FIFO에 비해 빠른 응답시간을 보이고 있다. 이러한 결과는 제안한 WCS가 동일 커넥션으로 요청된 전체 문서의 크기가 가장 작은 커넥션을 먼저 처리하고 스케줄링 윈도우를 사용으로 스케줄링의 범위가 제한되기 때문에 상대적으로 크기가 작은 커넥션이 먼저 처리되어 다른 두 스케줄링 기법에 비해 커넥션에 대한 응답 시간이 향상되게 나타난다. 따라서 문서의 크기를 고려하는 WCS는 문서의 크기 차가 크고, 한 커넥션을 통해 여러 개의 문서를 요청하는 경우에 우수한 성능을 보임을 알 수 있다.

위의 실험 결과들을 통해 본 논문에서 제안한 WCS는 문서의 크기에 따라 처리하는 SRPT-CS와 문서의 크기를 고려하지 않는 FIFO에 비해 우수한 결과가 나타났다. 이것은 크기가 작은 문서들을 먼저 처리하는



(a) 문서의 분포 : 0.5~9KB (b) 문서의 분포 : 0.07~858KB

그림 7 클라이언트의 증가에 따른 전체 커넥션에 대한 평균 응답시간

SRPT-CS는 같은 커넥션으로 동시에 요청된 문서들을 처리할 때 스케줄링의 범위가 모든 커넥션으로 확장될 수 있지만, SRPT-CS의 경우와 마찬가지로 문서의 크기를 기준으로 하는 WCS는 요청된 문서들의 크기로 정렬된 큐에 스케줄링 윈도우를 사용함으로써 스케줄링의 범위가 SRPT-CS에 비해 작기 때문에 SRPT-CS에 비해 빠른 응답시간이 나타났다. 그리고, FIFO는 문서의 크기와 상관없이 들어온 순서대로 처리를 하고 각 프로세스는 서버에서 운용되는 Unix와 같은 O/S 스케줄러의 영향을 받기 때문에 RR에 의해서 활성화가 된다. 따라서 스케줄링의 범위가 전체 커넥션으로 확장되어 크기가 작은 문서들의 처리가 지연될 수 있으며, 커넥션의 수가 증가할수록 서버의 부하가 증가하기 때문에 커넥션의 처리시간이 지연된다. 반면 WCS는 문서의 크기를 고려하고 스케줄링 윈도우를 사용함으로써 스케줄링의 범위가 작기 때문에 커넥션에 응답시간이 FIFO보다 우수하게 나타났다. 따라서 위의 실험결과로 문서의 크기를 고려한 WCS가 스케줄링 윈도우의 사용으로 기존에 제안된 SRPT-CS와 전통적인 스케줄링 방식인 FIFO보다 커넥션별 응답시간이 우수하다는 것을 볼 수 있었다.

5. 결 론

본 논문은 동시에 여러 문서를 요청할 수 있는 *HTTP/1.1* 프로토콜 환경에서 커넥션의 전체 응답시간을 향상시키는 스케줄링 윈도우를 이용한 커넥션 스케줄링 기법을 제안하였다. 제안한 WCS는 커넥션별로 요청된 문서들의 크기를 이용한 크기 기반의 스케줄링 기법으로 처리시간이 가장 짧게 남은 커넥션을 먼저 처리하는 SRPT를 적용한 커넥션 스케줄링 기법(SRPT-CS)과 비슷하다. SRPT-CS는 커넥션당 하나의 문서만 요청할 경우 커넥션에 대한 응답시간의 향상을 가져왔다. 그러나 pipelining을 통해서 동일 커넥션으로 동시에 상이한 크기의 문서를 요청 받을 경우, 논문에서 제안한 WCS에 비해 커넥션에 대한 응답시간이 저조하였다. 본 논문에서 제안된 WCS는 커넥션별로 요청된 전체 문서들의 크기를 고려하여 크기가 작은 커넥션을 먼저 처리하기 때문에 커넥션에 대한 응답시간이 향상되었다. 그리고, 스케줄링의 범위를 제한하는 스케줄링 윈도우의 사용으로 스케줄링의 범위가 제한되기 때문에 SRPT-CS와 FIFO보다 커넥션에 대한 응답시간의 개선이 가능했다.

본 논문에서 제안하는 WCS의 성능 평가를 위해 SRPT-CS와 FIFO의 전체 커넥션에 대한 응답 시간을 비교하였다. WCS는 요청된 각 문서들의 크기를 고려하는 SRPT-CS와 문서의 크기를 고려한다는 점이 같지만 스케줄링의 범위를 제한하는 스케줄링 윈도우를 사용하여 크기가 작은 커넥션을 먼저 처리한다는 점에서 SRPT-CS와 차이가 있다. 그리고, FIFO는 문서의 크기를 고려하지 않는 점에서 차이가 있다. 성능 평가의 결과를 통해 문서의 크기를 고려해서 크기가 작거나 처리되고 남은 양이 적은 문서를 먼저 처리하는 SRPT-CS가 요청된 문서들을 처리하는 모든 프로세스로 스케줄링의 범위가 확장된 FIFO에 비해 커넥션에 대한 응답시간이 우수하다는 것을 알 수 있었다. 그러나, SRPT- CS에 비해 본 논문에서 제안하는 WCS가 전체 커넥션에 대한 빠른 응답 시간을 보였는데, 이는 WCS가 SRPT-CS처럼 문서의 크기를 고려하지만 커넥션 단위의 크기를 고려하고 스케줄링의 범위를 제한하는 스케줄링 윈도우를 사용하여 크기가 작은 커넥션들을 먼저 처리하기 때문이다. 따라서 본 논문에서 제안한 스케줄링 윈도우를 사용하는 커넥션 스케줄링 기법이 전체 커넥션에 대한 응답시간에 있어서 기존 스케줄링 기법인 SRPT-CS와 FIFO보다 향상되었음을 보여주

었다.

본 논문에서 커넥션 스케줄링 기법 제안 시 동적 문서에 관한 부분을 고려하지 않았다. 문서의 크기와 처리 시간이 비례하지 않는 동적 문서에 대한 효율적인 스케줄링 기법과 웹서버의 전력을 고려한 스케줄링 기법은 향후 과제로 남긴다.

참 고 문 헌

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. "Hyper-Text Transfer Protocol-HTTP/1.1," IETF RFC 2616, June 1999.

[2] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection Scheduling in Web Servers," Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems, pp.243-254, October 1999.

[3] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal, "Size-Based Scheduling to Improve Web Performance," ACM Transactions on Computer Systems, Vol.21, No.2, pp.207-233, May 2003.

[4] E. J. Friedman and S. G. Henderson, "Fairness and Efficiency in Web Server Protocols," Proceedings of ACM SIGMETRICS'03, pp.229-237, June 2003.

[5] D. Lu, P. Dinda, Y. Qiao, and H. Sheng, "Effects and Implications of File Size/Service Time Correlation on Web Server Scheduling Policies," Technical Report NWU-CS-04-33, April 19, 2004.

[6] D. Lu, H. Sheng, and P. Dinda, "Size-based Scheduling Policies with Inaccurate Scheduling Information," Proceedings of the IEEE Computer Society's 12th Annual International Symposium on MASCOTS'04, pp.31-38, Oct. 2004.

[7] N. Bhatti and Rich Friedrich, "Web Server Support for Tiered Services", HPL-1999-160, 1999.

[8] J. Almeida, M. Dabu, A. Manikutty, and P.Cao, "Providing Different Levels of Service in Web Content Hosting," Proceedings of the Internet Server Performance Workshop, March 1998.

[9] P. Bhoj, S. Ramanathan, and S. Singhal, "Web2K: Bringing QoS to Web Servers," HPL-2000-61, 2000.

[10] M. Bender, S. Chakravarti, and S. Muthukrishnan, "Flow and Stretch Metrics For Scheduling Continuous Job Streams," Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, pp.270-279, 1998.

[11] C. Lu, T. Abdelzaher, J. Stankovic, and S. Son, "A Feedback Control Approach for Guaranteeing Relative Delays in Web Servers," IEEE Real-Time Technology and Applications Symposium, TaiPei, Taiwan, June 2001.

[12] The Apache Group. http://www.apache.org

[13] V. N. Padmanabhan and J. Mogul, "Improving HTTP Latency," Computer Networks and ISDN Systems, vol.28, pp.25-35, December 1995.

[14] V. S. Pai, P. Druschel, and W. Zwaenepoel, "Flash: An Efficient and Portable Web Server," Proceedings of the USENIX 1999 Annual Technical Conference, pp.199-212, June 1999.

[15] P. Barford and M. Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," Proceedings of Performance '98/ACM SIGMETRICS '98, pp.151-160, Madison WI. Slightly expanded version appears as BUCS-TR-1997-006, November 4, 1997.

[16] P. Barford, A. Bestavros, A. Bradley, and M. Crovella, "Changes in Web Client Access Patterns: Characteristics and Caching Implications," World Wide Web, Special Issue on Characterization and Performance Evaluation, Vol. 2, pp.15-28, 1999.

방 지 호

1997년 홍익대학교 컴퓨터공학과 졸업공학사). 2001년 홍익대학교 컴퓨터공학과 졸업(공학석사). 2001년 7월~현재 한국정보보호진흥원(KISA) 주임연구원. 2004년 3월~현재 홍익대학교 컴퓨터공학과 박사과정. 관심분야는 모바일/무선 컴퓨팅, 실시간 시스템 및 임베디드 시스템, 네트워크 및 시스템 보안

하 란

1987년 서울대학교 컴퓨터공학과 졸업 1989년 서울대학교 컴퓨터공학과 석사 1989년 3월~1990년 7월 한국통신 전임연구원. 1995년 University of Illinois at Uubana-Champaign 전산학 박사. 1995년 9월~현재 홍익대학교 컴퓨터공학과 부교수. 관심분야는 모바일/무선 컴퓨팅, 실시간 시스템, SDR, 네트워킹, 멀티미디어 시스템