HTTP/1.1 환경에서 응답시간을 개선한 정적 문서 스케줄링

방 지 호*, 하 란
홍익대학교 컴퓨터공학과
{jhbang, rhanha}@cs.hongik.ac.kr

Static Document Scheduling with Better Response Time in HTTP/1.1

Ji-ho Bang*, Rhan Ha
Dept. of Computer Engineering, HongIk University

요 약

접속이 빈번한 웹사이트들의 서버는 동시에 수백개의 커넥션을 처리해야 하므로 효율적인 커넥션 스케줄링 기법이 요구된다. 기존의 SRPT(Shortest Remaining Processing Time first)를 이용한 커넥션 스케줄링 기법은 가장 짧은 처리시간이 남아있는 커넥션을 먼저 처리함으로써 각 문서에 대한 응답시간의 향상을 가져왔다. 그러나, 클라이언트의 요청에 의해 형성된 하나의 커넥션으로 다수의 정적 문서들을 동시에 요청할 수 있는 HTTP/1.1에서 크기가 다른 다수의 정적 문서들이 요청됐을 때, 각 문서에 대한 응답시간은 빠를 수 있으나 커넥션에 대한 전체 응답시간의 향상은 보장할 수 없다. 따라서 본 논문은 HTTP/1.1 환경에서 웹 서버의 CPU와 메모리의 부하가 높을 때 발생되는 성능저하의 단점을 보완하면서 정적 문서 요청에 관한 응답시간을 향상시키는 pipelining 기반의 커넥션 스케줄링 기법을 제시한다. 실험을 통해서 제안한 커넥션 스케줄링 기법이 전체적으로 커넥션에 대한 빠른 응답시간을 보이고, 스케줄링 윈도우의 사용으로 스케줄링에 공평성을 보임을 알 수 있다.

1. 서 론

인터넷의 급격한 성장으로 웹서버는 동시에 많은 수의 커넥션(connection, 클라이언트의 요청으로 해당 서버와 형성되는 연결 또는 접속)을 처리하게 되었다. 그러나, 웹 서버가 동시에 수용할 수 있는 커넥션의 수는 한정되어 있으며 그러한 커넥션의 처리는 웹서버의 성능과 연관된다. 따라서 커넥션에 대한 효율적인 스케줄링 기법이 필요하게 되었다. 기존 연구된 기법으로는 SRPT(Shortest Remaining Processing Time first)를 이용한 커넥션 스케줄링(이하 SRPT-CS)[1]이 있다. 이 기법은 디스크 큐와 네트워크 큐에 저장된 커넥션 디스크립터의 정보와 현재 커넥션이 서비스 받아야할 데이터의 양에 따라 스케줄링을 하는 기법으로 크기가 작은 정적 문서에 대한 요청을 먼저 처리하기 때문에 각 문서에 대한 응답시간이 향상되었다. 그러나, 한 커넥션에 상이한 크기의 여러 정적 문서가 동시에 요청되었을 경우, 커넥션별 응답시간의 향상은 보장할 수 없다. 요청된 정적 문서들이 크기별로 정렬되었을 때, 크기의 차이가 큰 두 정적 문서를 동시에 요청한 커넥션은 다른 커넥션으로부터 요청된 크기가 작은 정적 문서에 의해 처리가 지연될 수 있기 때문이다. 따라서 커넥션 당 하나의 정적 문서만 요청 받아 처리하는 HTTP/1.0 서버에 SRPT-CS를 적용하였을 경우, 각 커넥션에 대해서 빠른 응답시간을 나타낼 수 있다. 그러나 동시에 여러 정적 문서들을 요청할 수 있는 HTTP/1.1의 pipelining을 고려해 보자. 한 커넥션으로 동시에 요청된 정적 문서들이 전부 처리된 시점을 해당 커넥션에 대한 처리가 끝난 시점으로 볼 때, pipelining이 적용되는 HTTP/1.1 서버에 SRPT-CS를 적용하였을 경우 커넥션에 대한 전체 응답시간의 향상을 보장할 수 없다.

위의 HTTP/1.0과 HTTP/1.1의 차이점은 persistent connection과 pipelining에 있다[2, 3]. HTTP/1.0 서버는 keep alive 인수의 설정에 의해서 persistent connection이 선택적으로 제공되었지만 불안정하였다. 그러나, HTTP/1.1 서버는 안정된 persistent connection이 기본적으로 제공된다[2]. 또, 클라이언트가 HTML 문서 내에 포함된 정적 문서들을 요청할 때, HTTP/1.0 서버와의 커넥션은 클라이언트가 요청하고자하는 정적 문서의 수만큼 형성되어 처리된다. 그러나, HTTP/1.1 서버에서는 클라이언트와 연결된 단 하나의 커넥션으로 여러 개의 정적 문서 요청과 응답이 가능하며, pipelining을 통해서 HTML 문서 내에 포함된 정적 문서들을 동시에 요청하고 제공받을 수 있다. 이때, pipelining을 통해서 동시에 요청된 정적 문서들의 응답순서는 반드시 요청된 순서와 같아야 한다[2]. 그리고, HTTP/1.1은 persistent connection을 기본적으로 제공하기 때문에 HTML 문서를 요청하기 위해 형성되었던 커넥션을 통해서 다른 정적 문서들도 요청할 수 있다. 따라서 HTTP/1.1에서 기본적으로 제공되는 persistent connection과 pipelining으로 커넥션 형성에 필요한 패킷의 수와 시간이 줄어들었다. 그러나 HTTP/1.1 서버는 HTTP/1.0 서버보다 커넥션을 유지하는 시간이 길기 때문에 CPU와 메모리의 부하가 높을 경우 HTTP/1.0 서버에 비해서 성능이 저하되는 것을 볼 수 있다[4]. 따라서 본 논문은 커넥션에 대한 빠른 응답시간과 제한적 공평성을 제공하면서 서버의 성능을 최적의 상태로 유지시키는 커넥션 스케줄링 기법을 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 HTTP/1.1 환경에서 개선된 응답시간을 나타내는 pipelining 기반의 커넥션 스케줄링 기법을 제안하고, 3장에서는 플래시(Flash) 웹 서버를 이용한 실험을 통해서

* 본 연구는 정보통신연구진흥원(과제번호 : 2000-202-01), 교육부(과제번호 : BK991031001)의 후원으로 연구되었습니다.

제안된 기법의 성능을 평가 분석한다. 그리고, 4장에서는 결론에 대해서 논한다.

2. Pipelining 기반의 커넥션 스케줄링 기법

HTTP/1.1 환경에서 클라이언트는 pipelining을 통해서 HTML 문서 내에 포함된 정적 문서들을 동시에 요청할 수 있으며, 요청한 순서대로 순차적으로 서비스 받을 수 있다. 그러나, persistent connection이 기본적으로 제공되기 때문에 사용되지 않는 커넥션은 바로 닫히지 않고 일정시간 동안 유지된다. 따라서 서버의 CPU와 메모리의 부하가 높을 때 성능이 저하되는 단점이 있다. 여기서는 본 논문에서 제안하는 pipelining 기반의 커넥션 스케줄링(Pipelining based Connection Scheduling, 이하 PCS) 기법에 대해서 보도록 하자. 먼저 persistent connection에 의해서 발생할 수 있는 성능저하의 단점을 보완한 PCS의 커넥션 종료시점을 보자. 그리고, HTTP/1.1 환경에서 개선된 응답시간이 나타나는 PCS의 알고리즘과 예제를 보도록 한다.

2.1 PCS(Pipelining based Connection Scheduling)

HTTP/1.1은 persistent connection이 기본적으로 제공되기 때문에 HTTP/1.0에 비해서 커넥션의 유지시간이 길다. 기존의 실험결과[4]로 HTTP/1.1 서버는 HTTP/1.0과 HTTP/1.1 early close보다 성능이 급격히 나빠지는 것을 볼 수 있다. 여기서, HTTP/1.1 early close는 pipelining 이용해서 GETALL[4,5]을 처리한 후 바로 커넥션을 닫기 때문에 HTTP/1.1 persistent connection 보다 커넥션 유지의 부담이 줄어든다. 그리고, pipelining을 통해서 처리하기 때문에 HTTP/1.0보다 적은 패킷이 사용된다. 따라서 본 논문의 PCS는 HTTP/1.1의 early close를 이용한다. 그리고, PCS의 알고리즘은 다음과 같다. 클라이언트의 요청에 따라 형성된 커넥션으로 HTML 문서 내에 포함된 정적 문서들이 동시에 요청되면, 커넥션 별로 요청된 정적 문서들의 크기(bytes)에 따라 커넥션의 정보를 큐에 정렬한다. 그리고, 스케줄링의 범위가 되는 스케줄링 윈도우의 크기(Scheduling Window Size, 이하 SWS)를 설정한다. 이때, SWS는 스케줄링이 시작되는 커넥션을 기준으로 해서 해당 커넥션으로 요청된 정적 문서의 수로 설정된다. 따라서 큐의 처음에 위치한 커넥션으로 요청된 정적 문서의 수가 3이면 SWS는 3으로 설정된다. 그리고, 스케줄링 윈도우 범위 안에 속한 커넥션들은 라운드 로빈(Round-Robin, 이하 RR) 스케줄링 기법으로 처리한다. 이때, 스케줄링의 시작은 큐의 맨 처음에 위치한 커넥션이 되며, 큐의 맨 처음에 위치한 커넥션부터 SWS만큼 떨어진 곳에 위치한 커넥션까지 스케줄링이 적용되는 범위가 된다. 한번의 read/write로 요청된 정적 문서를 처리할 수 있다고 가정하면, RR에 의해서 스케줄링 윈도우 범위의 처음에 위치한 커넥션의 첫 번째 정적 문서 요청을 처리하고, 다음 커넥션의 첫 번째 정적 문서를 처리한다. 윈도우의 끝까지 처리를 했으면 다시 맨 처음 커넥션의 두 번째 정적 문서를 처리한다. 처리 중 윈도우 범위 내에서 첫 번째 커넥션이 아닌 다른 커넥션이 먼저 끝날 경우 윈도우 범위 밖의 다른 커넥션을 윈도우 범위 안의 맨 뒤에 위치시키고 스케줄링을 한다. 그리고 첫 번째 커넥션의 정적 문서에 대한 요청이 다 처리되면 다음 커넥션의 처리되지 않은 요청의 수에 따라 스케줄링 윈도우 크기를 설정하고 스케줄링을 한다. 자세한 알고리즘은 표 1에 나와 있다.

```
PROCEDURE MultiRequest_Connection_Scheduling
/* 큐의 처음에 위치한 커넥션의 정보를 넘겨줌 */
connection_descriptor=first_connection_descriptor;
/* 스케줄링 윈도우 크기 설정 */
window_size=connection_descriptor.request_num;
for (j=0;j<window_size;j++)
{
  for (i=0;i<window_size;i++)
  {
    /* 해당 커넥션에서 요청된 정적 문서를 서비스 */
    open(connection_descriptor.file)
    read(connection_descriptor.file);
    write(connection_descriptor.file);
    /* 현재 처리되어야 할 문서의 수 */
    connection_descriptor.request_num--;
    if (connection_descriptor.request_num==0)
    { /* 윈도우 내의 커넥션 중 하나가 다 처리된 경우
       * 다 처리가된 커넥션이 가리키고 있는 다음 커넥션을
       * 처리가 다된 커넥션의 전 커넥션에게 포인터를 넘겨줌 */
      before_connection_descriptor.next->connection_descriptor=
      connection_descriptor.next->connection_descriptor;
      close(connection_descriptor.connection_id);
    }
    else
    { /* 다음 처리될 정적 문서에 대한 포인터와
       * 다음 커넥션에 대한 포인터를 넘겨줌 */
      connection_descriptor.file=connection_descriptor.file->file;
      connection_descriptor=
      connection_descriptor.next->connection_descriptor;
    }
  }
  connection_descriptor=first_connection_descriptor;
  }
END PROCEDURE
```

표 1 PCS 알고리즘

2.1.1 PCS의 예

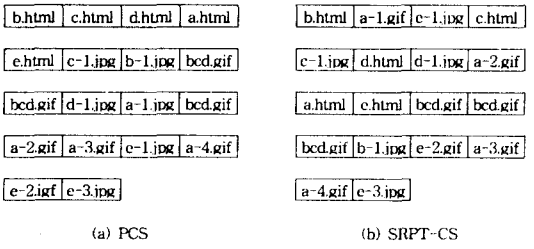
여러 개의 정적 문서들을 포함하고 있는 5개의 HTML문서들로 구성되어 있는 웹사이트가 있다고 가정하자. 각 정적 문서의 크기는 표 2와 같으며, 문서의 크기와 상관없이 한번의 read/write로 처리할 수 있다고 하자. 그리고, HTTP/1.1 서버는 클라이언트에 의해 요청된 순서대로 처리해서 클라이언트에게 응답해야 한다. 따라서, 기존에 제안된 SRPT-CS가 HTTP/1.1의 제약을 위반하지 않는 범위에서 본 논문의 PCS와 커넥션에 대한 응답시간을 비교하기 위해 각 HTML문서에 포함되어 있는 정적 문서들의 크기를 임의로 오름차순으로 정렬하였다. 그리고 정렬된 순서대로 클라이언트에 의해서 요청한다고 하자. a.html문서 내에 포함된 정적 문서들을 a-class(a-1.gif, a-2.gif, a-3.gif, a-4.gif), b.html문서 내에 포함된 정적 문서들을 b-class(b-1.jpg, bcd.gif), c.html문서 내에 포함된 정적 문서들을 c-class(c-1.jpg, bcd.gif), d.html문서 내에 포함된 정적 문서들을 d-class(d-1.jpg, bcd.gif) 그리고, e.html문서 내에 포함된 정적 문서들을 e-class(e-1.jpg, e-2.gif, e-3.jpg)라고 하자. 그리고, 임의의 t 시간에 A, C, E, G, I 클라이언트들은 GET method로 각각의 a.html(3177), b.html(102), c.html(1774), d.html(2153), e.html(3638) 문서들을 요청했으며, 먼저 각각의 HTML문서를 요청했던 B, D, F, H, J 클라이언트는 각각의 HTML문서를 파싱(parsing)한 후 HTML문서 내에 포함된 a.class(22878), b.class(15671), c.class(1774), d.class(16019), e.class(156134)의 이미지 문서들을 요청했다. 그리고 PCS 알고리즘에 의해서 각각의 커넥션들을 요청한 정적 문서들의 전체 크기로 정렬한 클라이언트의 순서는 그림 1과 같다. 큐의 앞에서부터 C, E, G, A, I 클라이언트에 의해서 한 개의 HTML문서가 요청되었기 때문에 PCS알고리즘에 의해서 SWS가 1로 설정되고, C, E, G, A, I 클라이언트 순서로 스케줄링 된다. C 클라이언트의 b.html 문서가 처리되고, 계속 I 클라이언트의 e.html 문서가 처리될 때까지 SWS는 1인 상태로 해서 각각의 HTML문서가 처리된다. 그 다음에 위치한 F, D, H, B, J 클라이언트는 요청한 문서의 수가 2이상이기 때문에 계속해서 SWS가 변하게 된다. 큐의 앞쪽에 위치한 F 클라이언트에서 2개의 정적 문서(b-1.jpg, bcd.gif)가 요청되었기 때문에 PCS 알고리즘에 의해서 SWS는 2로 설정되고 스케줄링의 범위는 F부터 D 클라이언트까지가 되며, RR 스케줄링 기법으로 F와 D 클라이언트에서 요청된 정적 문서들을 처리한다. 그리고 F 클라이언트에서 요청된 정적 문서들이 다 처리됨과 동시에 D 클라이언트에서 요청된 정적 문서도 다 처리되기 때문에

문서	크기(Byte)	문서	크기(Byte)	문서	크기(Byte)	문서	크기(Byte)
a.html	3177	a-4.gif	39358	c-1.jpg	1956	e.html	3638
a-1.gif	218	b.html	102	d.html	2153	e-1.jpg	1356
a-2.gif	2234	b-1.jpg	15671	d-1.jpg	2223	e-2.gif	10703
a-3.gif	20426	c.html	1774	bcd.gif	13786	e-3.jpg	135075

표 2 각 정적 문서들의 크기

C	E	G	A	I	F	D	H	B	J

그림 1 큐에 저장된 클라이언트의 순서



(a) PCS (b) SRPT-CS

그림 2 정적 문서 처리순서

다음에 위치한 H 클라이언트로 스케줄링 시점이 넘어가게 된다. H 클라이언트에서 2개의 정적 문서가 요청되었기 때문에 SWS는 2로 설정되어 B 클라이언트까지 스케줄링의 범위가 된다. PCS 알고리즘에 의해서 H 클라이언트에서 요청된 정적 문서가 다 처리되었을 때 B 클라이언트에서 요청된 정적 문서는 4개 중 2개의 정적 문서(a-1.gif, a-2.gif)가 처리되고 2개의 정적 문서(a-3.gif, a-4.gif)가 남게 된다. 그래서, SWS는 H 클라이언트에서 요청된 정적 문서 중 처리가 남은 2개의 정적 문서의 수만큼 설정(SWS=2)되어 B 클라이언트부터 J 클라이언트까지가 스케줄링의 범위가 되어 RR 스케줄링 기법에 의해서 처리된다. 따라서 PCS 알고리즘에 의한 전체적인 스케줄링은 그림 2(a)와 같다. 그러나 각 문서의 크기에 따른 스케줄링 기법인 SRPT-CS으로 처리했을 때 스케줄링 되는 순서는 그림 2(b)와 같다.

다음 장에서본 논문에서 제안한 PCS와 SPRT-CS의 응답시간을 비교해 보도록 하자.

3. 성능 평가

널리 사용되고 있는 아파치(Apache)[6]와 Rice 대학에서 개발한 플래시(Flash)[7] 웹서버는 코드가 공개되어있다. 그러나 이 두 서버는 서로 동작하는 방식이 다르다. 아파치는 멀티프로세스 방식으로 동작을 하지만 플래시는 단일프로세스 방식으로 동작을 하면서 디스크 I/O 필요시 헬퍼(helper) 프로세스가 디스크 I/O를 담당하는 AMPED (Asymmetric Multi-Process Event Driven) 방식이다. 멀티프로세스 방식은 스케줄링의 어려움이 있는 반면 단일프로세스 방식은 스케줄링이 용이하다. 따라서 본 논문에서 제시한 PCS을 플래시 기반에 구현을 하였다. 그러나 PCS는 플래시가 자체적으로 가지고 있는 LRU 캐쉬와 Nonblocking I/O를 가능하게 하기 위해 사용한 헬퍼 프로세스의 특성을 이용하는 것이 아니기 때문에 플래시 코드의 수정시 배제하였다. PCS의 성능을 평가하기 위해서 기존의 SRPT-CS와 비교를 하였다. 또, 아파치는 요청된 문서의 크기와는 상관없이 처리를 하는 size-independent 방식이지만, PCS는 size-dependent 방식이다. 따라서 size-independent 방식과도 비교를 하였다. PCS는 pipelining의 특성을 이용한 스케줄링 기법이기 때문에 모든 클라이언트는 GETALL로 정적 문서를 요청하도록 하였다. 그림 3은 실험에 사용된 정적 문서의 분포로 각 문서(x)에 대한 문서의 크기(y)를 나타낸다. 그리고 그림 4와 그림 5는 동시에 형성된 커넥션으로부터 서로 다른 정적 문서의 요청이 들어 왔을 경우에 대한 실험결과이다. 그림 4는 각 커넥션(x)으로부터 요청 받은 서로 다른 정적 문서가 다 서비스된 후 시간(y)에 대한 결과로 PCS가 다른 스케줄링 방식에 비해 응답시간이 빠른 것을 볼 수 있다. 그리고, 그림 5는 각 커넥션(x)의 요청에 대한 서비스 지연시간(y)에 대한 결과로 PCS가 다른 스케줄링 방식에 비해 서비스를 받는 지연시간이 작으며, 각 커넥션에게 스케줄링의 공평한 기회를 제공하는 것을 볼 수 있다. 따라서 본 논문에서 제안한 PCS는 기존의 SRPT-CS과 size-independent한 방식과 비교해 볼 때 커넥션별 응답시간이 우수하며, 지연시간이 작은 것을 볼 수 있다.

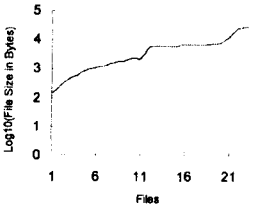


그림 3 정적 문서의 분포

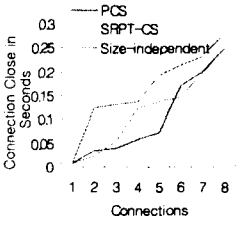


그림 4 커넥션에 대한 응답시간

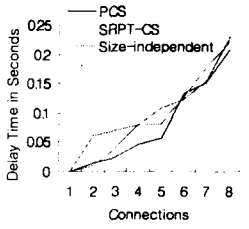


그림 5 커넥션에 대한 지연시간

4. 결 론

기존에 SRPT를 이용한 커넥션 스케줄링 기법은 정적 문서의 크기에 대한 스케줄링 기법으로 HTTP/1.0 환경에서 요청된 정적 문서에 대해 응답시간의 향상을 가져왔다. 그러나 동일한 커넥션으로 동시에 상이한 크기의 정적 문서를 요청 받을 수 있는 HTTP/1.1의 pipelining에 적용하였을 경우, 커넥션에 대한 응답시간의 향상을 보장할 수 없다. 따라서 본 논문에서는 HTTP/1.1 환경에서 응답시간을 개선한 정적 문서 스케줄링 기법인 pipelining 기반의 커넥션 스케줄링(PCS) 기법을 제안하였다. 본 논문에서 제안된 PCS는 커넥션별로 요청된 정적 문서들의 크기를 고려하였으며, SWS를 통해서 스케줄링의 범위를 제한하였다. 그리고, 커넥션 유지의 부담을 줄이기 위해서 커넥션 종료시점에 HTTP/1.1 early close를 적용하였다. 또, PCS의 성능을 평가하기 위해서 플래시 웹서버를 이용하였으며, 동일한 서버에 SRPT-CS와 Size-independent 방식의 스케줄링 기법을 구현한 후 성능을 비교하였다. 실험결과를 통해서 본 논문에서 제시한 PCS가 커넥션 응답시간 및 서비스 지연시간에 있어서 기존 커넥션 스케줄링 기법보다 향상되었음을 볼 수 있었다.

참고 문헌

[1] 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 (USITS '99), pp.243 254, Oct. 1999.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. Hypertext transfer protocol-HTTP/1.1 IETF RFC 2616, June 1999.

[3] H. F. Nielsen, J. Gettys, A. Baird-Smith, E. Prud'hommeaux, H. Lie and C. Lilley, "Network Performance Effects of HTTP/1.1, CSS1, and PNG", Proceedings of ACM SIGCOMM '97, pp.155-166, Sep. 1997.

[4] P. Barford and M. Crovella, "A performance Evaluation of Hyper Text Transfer Protocols", Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer systems, pp.188-197, May 1999.

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

[6] The Apache Group. Apache web server. http://www.apache.org

[7] 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.