우선 순위를 갖는 웹서버 컨넥션 스케쥴링

염미령 노삼혁
홍익대학교 컴퓨터공학과

Priority Connection Scheduling in Web Servers

Mi-ryeong Yeom Sam H. Noh
Dept. of Computer Engineering, Hong-Ik University

요 약

웹서버는 disk I/O 오버헤드를 줄이기 위해 웹 문서를 메인 메모리 캐싱 한다. 하지만 동시에 들어오는 connection들에 대해 처리 순서는 고려하지 않으므로 같은 문서를 요구하는 request들이 메모리 캐쉬의 working set내에 들어오지 않을 경우 disk I/O를 추가시킬 수 있다. 본 논문에서는 동시에 처리해야 할 같은 작업량을 요구하는 request는 연속적으로 처리함으로써 disk I/O 오버헤드를 줄이는 우선 순위 스케쥴링을 수행함으로써 정적 웹 환경에서의 사용자 응답 시간을 줄였다.

1. 서 론

고성능 웹서버는 동시에 수 백개 이상의 connection들을 처리할 수 있으며 이들의 효율적인 처리를 위해 메인 메모리 캐싱을 한다. 그러나, 대부분의 웹서버는 동시에 수행해야할 connection들의 수행 순서를 OS에 의존하며 웹 서버 내에서 특별한 스케쥴링을 하지는 않는다. 만약, 클라이언트가 요청하는 문서에 대한 정보를 미리 알 수만 있다면, 동시에 들어오는 connection들을 스케쥴링할 수 있으므로 기존의 OS에 의존적인 처리 방식보다는 웹서버의 성능을 높일 수 있다.

정적 웹 환경에서는 파일 시스템과는 달리 문서 전체를 요구하게 되므로, 서버의 서비스 시간은 문서의 크기에 비례한다. 즉, 크기가 작은 문서부터 서비스할 경우 사용자의 평균 응답 시간이 줄어들게 됨을 의미한다. 또 동시에 들어오는 수십 내지 수백 개의 connection들에 대해 같은 작업을 요구하는 connection들을 연속적으로 서비스한다면 disk I/O의 횟수를 줄일 수 있는 가능성이 높아져 사용자 응답시간을 줄이게 된다.

Crovella[2]는 웹서버가 처리하는 connection들을 SRPT (Shortest Remaining Processing Time first) 스케쥴링으로 웹서버의 성능을 향상시켰다. SRPT 스케쥴링은 운영체제의 CPU스케쥴링에 제안되었던 방법으로 프로세스의 수행시간을 미리 알아야만 하고, 수행시간이 큰 프로세스는 기아(starvation)를 일으킬 가능성이 많기 때문에 이론적인 스케쥴링 방법으로만 알려졌다.

그러나, 정적 웹 환경에서는 클라이언트가 요구하는 HTTP request들의 사이즈를 미리 알 수 있으므로 SRPT 스케쥴링 방식을 적용시킬 수 있다. 또, HTTP request들의 분포가 heavy-tail (적은 수의 매우 긴 connection이 처리하는 작업양은 전체 connection 작업량의 많은 부분을 차지[3])이기 때문에 긴 connection들에게 크게 영향을 미치지 않고도 size-independent 스케쥴링에 비해 응답 시간을 향상시켰다.

본 논문에서는 수행해야 할 작업이 작은 파일과 동일한 작업량을 요청하는 connection에 우선 순위를 주는 스케쥴링 코드를 FLASH[1] 웹 서버 모듈에 삽입하여 실험한 결과, 고부하(high load)일 때 사용자의 응답 시간을 50% 이상 줄였다.

본 논문의 구성을 살펴보면, 2절에서는 기존 웹서버에서의 구동 방식에 대해 정리하고 3절에서는 플래쉬 웹서버의 connection 수행과 문제점에 대해 알아보며, 4절에서는 본 논문에서 제시하는 스케쥴링에 대해 소개를 할 것이다. 5절에서 실험 결과를 보여주고 6절에서는 결론 및 향후 연구 과제에 대해 논의한다.

2. 웹서버 구동 방식

기존 웹서버들의 구동 방식은 크게 MP, SPED, AMPED 등의 3가지 방식으로 나뉜다.

1) MP (Multiple-Process)

MP또는 MT(Multiple-Thread)방식의 웹 서버는 새로운 connection들이 들어올 때마다 프로세스(또는 스래드)를 생성한다. 들어온 여러 개의 connection들은 각각의 프로세스(스래드)에서 수행된다. 프로세스(스래드)의 스케쥴링은 OS의 CPU 스케쥴링을 따르므로 웹서버는 connection 스케쥴링에 관여하지 않는다.

2) SPED (Single-Process Event Driven)

SPED 방식의 웹서버에서는 모든 HTTP request의 수행을 하나의 프로세스가 처리한다. Zeus 웹서버[4], Squid[5] 프락시 서버에서 사용되는 방식으로 SPED는 단일 프로세스 처리 방식이다. 문맥전환(context switch)의 오버헤드가 없기 때문에 MP보다 성능이 우수한 것으로 알려졌다. 그러나, disk I/O 수행중에 프로세스가 blocking되는 단점이 있다.

3) AMPED (Asymmetric Multiple-Process Event Driven)

APMED 방식의 웹서버는 캐쉬되어 있는 작업량에 대해서는 SPED 방식으로 수행하다가 캐쉬되어 있지 않은 작업량을 처리하기 위해서는 MP 방식으로 구동한다. 모든 HTTP request의 수행은 하나의 프로세스가 처리하는 SPED 방식의 단점을 극복하는 방식으로 플래쉬 웹서버(FLASH)가 이 방식을 취한다[1]. 플래쉬는 캐쉬된 작업량들에 대해서는 SPED로 동작하고 캐쉬되지 않은 작업량을 수행해야하는 시점에서 헬퍼 프로세스 (helper process)들이 disk I/O를 대신하므로, 디스크 접근시 SPED가 블록되는 지연을 막아준다. 플래쉬 웹서버에서도 특별한 connection 스케쥴링은 하지 않으며 OS에 의존한다.

3. 플래쉬 웹서버에서의 connection 수행에서의 문제점

플래쉬 웹서버는 시스템이 제공하는 최대 connection 수의 크기로 구성된 read 비트맵 테이블과 write 비트맵 테이블을 유지한다. 새로 들어오는 connection들은 FD(File Descriptor)를 할당받고 비트맵 테이블의 빈 슬롯을 찾아 할당한다. Select (또는 poll) 함수는 정해진 timeout내에 셋팅된 각 비트맵들을 스캔한 후 어느 FD에서 event가 발생했는지를 탐지한다. 그 다음, 애플리케이션(플래쉬 웹서버)에서는 Select 함수가 탐지해 놓은 각 비트맵들을 한번에 오름차순으로 스캔하면서 event에 적절한 핸들러 루틴을 수행한다. 이런 방식에서는 먼저 들어온 connection이 먼저 수행된다는 보장이 없다. 늦게 들어온 connection이라 할지라도 비트맵 테이블의 작은 번호 슬롯을 할당받으면, 먼저 수행되므로 작업이 빨리 완성될 가능성이 높다.

새로운 connection이 들어오면 비트맵의 빈 슬롯을 할당받는다. 초기에는 빈 슬롯이 많으므로, 작은 번호순으로 할당 받게 되나 시간이 흘러 들어오는 connection의 수가 많아지면, 작업을 마치고 반납한 빈 슬롯은 점점 sparse하게 남는다. 비트맵 테이블이 매우 클 때는 빈 슬롯이 더욱 더 sparse하게 생기므로, 동시에 같은 문서를 요구하는 connection들이 비트맵 테이블내의 슬롯 거리 간격을 아주 멀게 할당받을 수 있다. 이런 경우, 추가적인 disk I/O 오버헤드가 생기게 된다.

예를 들어 그림 1에서와 같이, 4번 슬롯과 N-2번 슬롯이 같은 작업량을(workload) 요구하는 connection이고 캐쉬의 working set은 (2,10)이라면, N-2 슬롯의 request가 수행될 때는 해당 문서가 메모리에 존재하지 않아 disk access를 해야한다.

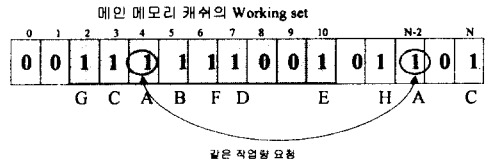

그림 1. EVENT 비트맵

4. 플래쉬 웹서버에서 컨넥션 스케쥴링의 제안

한번에 수행해야 할 connection들이 셋팅 된 비트맵을 스케쥴링 윈도우라 부르자. 본 논문에서 스케쥴링이라 함은 스케쥴링 윈도우 내에 셋팅되어 있는 connection들을 스케쥴링하는 것이다. SRPT[2]는 문서의 사이즈만을 고려해 스케쥴링하였으나 본 논문에서는 수행해야 할 작업량의 크기를 우선 고려하면서, 같은 작업량을 요구하는 connection을 연속적으로 수행함으로써 disk I/O의 빈도 수를 줄이고자 하였다. 본 논문에서 제안하는 스케쥴링 방식은 다음과 같다.

첫째, 작은 작업량을 가진 connection 우선 수행한다. 사용자의 request의 대부분은 작은 문서이며 문서의 요청 인기도는 Zipf[3] 분포를 따르므로 작은 사이즈의 문서를 요청하는 connection에 우선 순위를 부여한다. 즉, 처리해야할 작업량이 가장 적은 문서를 우선으로 수행한다(SRPT). 둘째, 스케쥴링 윈도우 내에서 같은 작업량을 요청하는 connection들은 연속적으로 수행한다.

같은 작업량에 대한 요청을 한꺼번에 묶어 연속적으로 수행하므로, 재 참조되는 데이터는 캐쉬 내에 존재할 확률이 아주 높다. 이것은 disk I/O를 횟수를 줄이게 되어 메모리 캐쉬의 힛트율(hit rate)을 높이며 사용자 응답시간을 단축할 수 있다. 또, 스케쥴링을 함으로써 disk I/O의 발생 빈도수를 줄이면, disk I/O를 수행하는 helper 프로세스의 생성

을 억제하게되므로 프로세스가 차지하는 메모리 공간의 낭비를 줄일 수 있으며, 프로세스간의 문맥전환 오버헤드를 줄일 수 있다.

아래의 그림2는 스케줄링 전의 슬롯 할당과 후의 슬롯 할당을 보여준다. 문서의 크기는 A<B<C<D<E<F<G<H 이다. 스케줄링을 하지 않는다면, 수행 순서는 비트맵의 오름차순인 G, C, A, B, F, D, E, H, A, C가 된다. 여기서 G, C, E, B, D를 수행 후 H,A,C를 수행하기 위해서는 disk I/O가 필요할 것이다. 그러나, 이들을 스케줄링하면, A, A, B, C, C, D, E, F, G, H의 순서로 수행된다. 같은 작업을 요청하는 connection들이 연속적으로 수행되므로 메인 메모리 캐쉬의 working set내에 들어올 확률이 아주 높아진다. 즉, A와C는 메인 메모리 캐쉬내에 존재하게 되어 disk I/O가 필요치 않을 수도 있다.

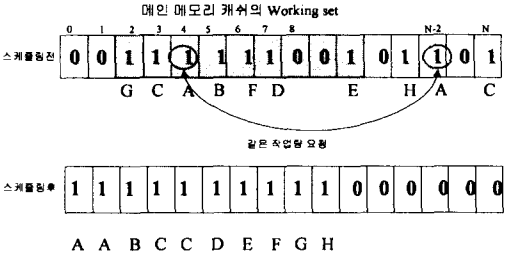

그림 2. 스케줄링 전/후의 event 비트맵

5. 실험 및 결과

실험한 웹서버들은 LINUX 2.2.16, 128M RAM, 팬티엄 II 350Mhz의 머신 위에서 구동되었으며, 2대의 클라이언트 머신은 128M RAM, Pentium III 750Mhz위에서 구동되었다. 클라이언트는 징적 connection들을 자동으로 생성해주는 SURGE[6]를 사용했다. 각 클라이언트 머신은 45 ~ 135개의 스래드(UE: User Equivalent)가 HTTP request들은 연속적으로 생성해 낸다. 모든 실험은 1분 동안 클라이언트를 연속적으로 생성시켰으며, 이것을 5번 반복하여 평균값을 나타낸 수치이다.

동시에 들어오는 connection의 수가 많지 않은 210 UE 미만에서는 플래쉬 웹서버와 스케줄링된 플래쉬 웹서버의 응답 시간은 크게 차이가 나지 않았으나 서버가 고부하(high load)가 되는 210UE 이상에서는 50%이상의 차이가 났음을 알 수 있다. 이것은 동시에 들어오는 connection들이 많을수록 스케줄링의 효과가 크다는 것을 보여준다.

플래쉬 웹서버는 고부하 상태에서 매우 불안정하여 다운되는 일이 잦았으나 스케줄링된 플래쉬 웹서버는 고부하 상태에서도 서버가 다운되는 일이 거의 없었다. 이것은 작은 파일을 우선으로 서비스하기 때문에 connection의 연결 지속 시간이 짧아지므로 원활한 FD 할당이 이루어지고, 같은 작업량을 접근하는 문서를 연속적으로 서비스해주어 disk I/O의 횟수를 줄이므로 적은 수의 helper로도 서비스가 가능하기 때문이다.

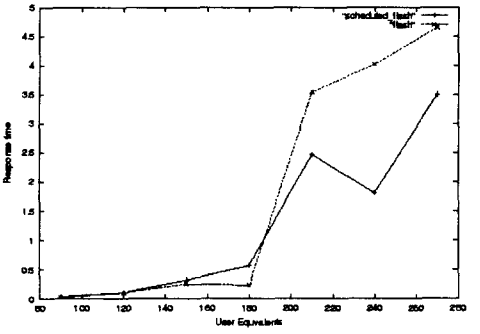

그림 3. 사용자 응답시간과 UE(User Equivalent)

6. 결론 및 향후 연구과제

본 논문에서는 동시에 들어오는 connection들을 스케줄링하여 disk I/O를 줄일 뿐만 아니라 시스템의 부하를 낮추어 서버의 처리율을 향상시킴으로서 사용자의 응답 시간을 줄였다. 본 논문에서 제안한 스케줄링 기법은 플래쉬 서버뿐만 아니라, 현재 70%이상의 웹서버 시장을 점유하는 MP 방식의 아파치 서버[7]에도 적용시킨다면 사용자 응답 시간을 줄일 수 있을 것이다. 앞으로, 본 논문에서 제안한 스케줄링 기법을 기반으로, 웹 프락시 서버에도 적용할 예정이다.

참고문헌

[1]Vivek Pai, Peter Druschel, and Willy Zwaenepoel. "Flash: An Efficient and Portable Web Server", In *Proceedings of the USENIX 1999 Annual Technical Conference*, Monterey, CA, June 1999.

[2]Mark E. Crovella, Robert Frangioso, and Mor Harchol-Balter. "Connection Scheduling in Web Servers", 2nd USENIX Symposium on internet Technologies and Systems.

[3]Mark E. Crovella, Murad S. Taqqu, and Azer Bestravros. Heavy-Tailed Probability Distributions in the World Wide Web. In A Practical Guide To Heavy Tails, chapter 1, pages 3-26. Chapman & Hall, New York, 1998.

[4]Zeus Technology Limited. Zeus Web Server. http://www.zeus.co.uk

[5]http://squid-cached.org

[6]Paul Barford and Mark Crovella. "Generating Representative Web Workloads for Network and Server Performance Evaluation", In Proceedings of SIGMETRICS '98, pages 151-160, July 1998.

[7]Apache. http://www.apache.org