이중 큐 구조를 갖는 웹 서버

염 미 령†

요 약

본 논문에서 구현한 더블큐 웹서버는 동시에 들어오는 요청들을 두 가지로 분류하여 서비스한다. 캐쉬되어 있는 문서를 요구하는 요청은 서비스 큐에 넣고, 캐쉬되지 않은 문서를 요구하는 요청은 기다림 큐에 넣는다. 더블큐 웹서버는 서비스 큐의 모든 문서를 서비스 한 후 기다림 큐의 요청을 서비스한다. 이 방식은 디스크 접근 오버헤드를 줄이기 위해 캐쉬 된 문서의 서비스를 우선하는 정책으로 아파치 웹서버와 비교 실험 결과 서버의 성능과 평균 사용자 응답 시간을 향상시켰다.

Double Queue Management for Reducing disk I/O of Web Servers

Miryeong Yeom†

ABSTRACT

This paper propose the DoubleQ web server that classifies incoming requests according to whether the requested document is cached or not. Requests that demand a cached document is put into the Service Queue while other requests are added to the Defer Queue. The DoubleQ web server services requests that are in the Service Queue before it services requests in the Defer Queue. This strategy is used to reduce disk accesses that have been the predominant overhead of traditional web servers. Experimental results using synthetic data show that improvements in the average user response time and the throughput of the web server may be achieved.

키워드 : 웹서버(web server performance), 컨넥션 스캐쥴링(connection scheduling)

1. 서 론

웹은 인터넷상에 광범위하게 분산된 다양한 형식의 데이터를 브라우저를 통해 손쉽게 검색할 수 있을 뿐만 아니라 모든 응용의 사용자 환경을 손쉽게 포함함으로써 새로운 데이터의 통합을 위한 가장 성공적인 기술로 평가받고 있다. 최근 이러한 웹의 고도화되고 다양한 기능으로 인해 그 사용자 수는 폭발적인 증가 추세로 2002년까지 3억 2천 여명에 달할 것으로 예상한다[1]. 그러나, 서버는 사용자의 요구에 대한 수요를 감당하지 못해 응답 시간이 길어짐과 동시에 사용자의 인내심을 넘어서게 되었다[2]. WWW 지연시간(latency)은 부하가 크지 않을 때는 요청 당 평균 3~4초이지만, 과부하가 발생할 경우에는 10초 이상이 걸린다[4, 5]. 그러나, [15]에서는 웹 사이트에서의 응답 시간은 1초가 바람직하다고 하였다. 이것은 인간이 감지하기에 적절한 속도이다. 웹 서버의 응답 시간이 길수록 인터넷 기업의 사용자들은 서버의 성능이 나쁘다라고 믿게 될 것이며 더욱이 보안(security) 부분까지도 의심하게 되므로 결국, 고객들은 자연히 이탈한다[3]. 대규모의 가입자를 보유하고 있는 다수의 인터넷 기업들은 사용자의 증가에 따라 서버의 용량 증가에 많은 비용을 투자하고 있는 실정이며 사용자의 요구를 만족시키기 위해 끊임없이 용량을 늘여야만 하는 상황에 이르렀다. 그러므로, 늘어나는 요청률에 응답하기 위해서도 웹서버는 불필요한 오버헤드는 피해야 한다. 정적 환경에서 웹 서버의 가장 큰 오버헤드는 디스크 액세스이다. 정적 환경이란 사용자 대부분이 HTTP GET 메소드를 통해 문서를 요청하는 것을 의미한다. 클라이언트가 웹 서버에게 문서를 요청하면, 웹 서버는 파일 시스템을 호출하여 파일을 open/read/write/close하게 된다. 이들 시스템 호출은 파일 시스템에 의존하며 하나 또는 그 이상의 디스크 접근을 하게 된다. 만약에 문서가 서버의 로컬 디스크에 존재하지 않는다면 네트워크 파일 시스템으로부터 가져와야 하는 아주 큰 오버헤드가 발생할 수도 있다. 그러므로, 가장 인기있는 문서들을 메모리에 캐쉬함으로써 시스템 호출의 오버헤드와 디스크 접근 횟수를 줄일 수 있다.

본 논문에서는 효율적인 웹 서비스를 제공하는 더블큐 웹서버를 제시한다. 더블 큐 웹 서버는 클라이언트의 요청을 도착 순서대로 서비스 해주지 않는다. 요구한 문서가 캐쉬에 없다면 서비스 기다림 상태가 되어 기다림 큐로 들어간다. 캐쉬

† 준 회 원 : 홍익대학교 대학원 전자계산학과
논문접수 : 2001년 10월 4일, 심사완료 : 2001년 11월 26일

되어 있다면 서비스 큐로 들어간다. 서비스 기다림이란 서버에 도착한 클라이언트의 요청이 캐쉬 미스를 발생시킬 때 서비스를 지연시키는 것을 의미한다. 이렇게 미스가 일어나는 서비스의 요구를 늦추게 되면 요구 순서와 관계없이 히트가 예상되는 문서들이 먼저 서비스될 수 있다. 그러므로, 가까운 미래에 참조될 문서를 캐쉬에서 쫓아내는 오류를 줄이게 된다. 또한 각 큐에서 대기중인 요청들은 같은 문서를 요구하는 요청들을 일괄적으로 서비스함으로써 사용하던 자원을 빨리 반납하게 한다. 본 논문에서는 캐쉬 미스를 유발시키는 요청들의 서비스를 지연시킴으로써 웹 서버의 성능을 향상시키는지에 대해 실험하기 위해 Event_Driven 방식의 웹 서버를 구현하여 아파치 웹 서버와 성능을 비교하였다. 본 논문에서 구현한 웹 서버는 저부하 일 때뿐만 아니라 고부하일 때에도 항상 아파치 웹 서버[10]보다 사용자 응답 시간을 향상시켰으며 웹 서버의 처리율도 향상되었다.

본 논문의 구성은 다음과 같다. 제 2절에서는 관련 연구에 대해 서술하며 제 3절에서는 예비실험에 대해서 언급하며 제 4절에서는 본 논문에서 제시하는 더블 큐에 대해 설명한다. 제 5절에서는 논문에서 사용된 실험 데이터에 대해서 설명한다. 제 6절에서는 실험 후 성능 평가를 그리고 마지막으로 제 7절에서 결론을 맺는다.

2. 관련 연구

캐쉬에 어떤 문서를 두느냐를 결정하는 것은 웹 서버의 성능을 크게 좌우 할 수 있다. 주 기억 장치의 크기가 서버의 디스크보다 상대적으로 작으므로 여러 문서들은 캐쉬를 두고 경쟁하게 된다. 캐쉬에 빈 공간이 없을 때 캐쉬에 없는 문서를 요구한다면 캐쉬 내에서 가장 쓸모 없는 문서는 쫓아내고 요구한 문서를 캐쉬로 가져와야 한다. 웹 문서는 시간적 지역성을 갖는다. 이것은 참조된 문서가 가까운 미래에 다시 참조되는 것을 의미하며 약 30%의 문서가 시간적 지역성을 띤다[6]. Markatos[7]는 메인 메모리 캐싱을 수행함으로써 웹 서버의 성능 향상을 보여주었으며 대부분의 고성능 웹 서버는 메인 메모리 캐싱을 수행한다[10-12]. 또, Pai[16]와 Aron[17]은 웹 문서의 요청들이 지역성을 갖는 성질을 이용하여 웹 클러스터링 시스템에 동적 로드 밸런싱을 적용하여 시스템의 성능을 향상시켰다.

대부분의 운영 체제 시스템은 가상 메모리 시스템이며 디스크의 파일은 블록 입출력 동작을 통해 버퍼캐쉬를 사용하던지 페이지 입출력 동작을 통해 페이지 캐쉬로 로드 된다. 각 동작의 장단점을 살펴보면 다음과 같다.

- 블록 입출력 동작의 사용
  - read 시스템 호출의 오버헤드 발생
  - 파일 시스템과 웹서버 사용자 공간의 데이터 중복 저장
- 페이지 입출력 동작의 사용
  - mmap 동작의 사용으로 read/write 시스템 호출 오버헤드 없음
  - 파일 시스템과 웹 서버 사용자 공간의 데이터 중복을 방지
  - 페이지 폴트 오버헤드

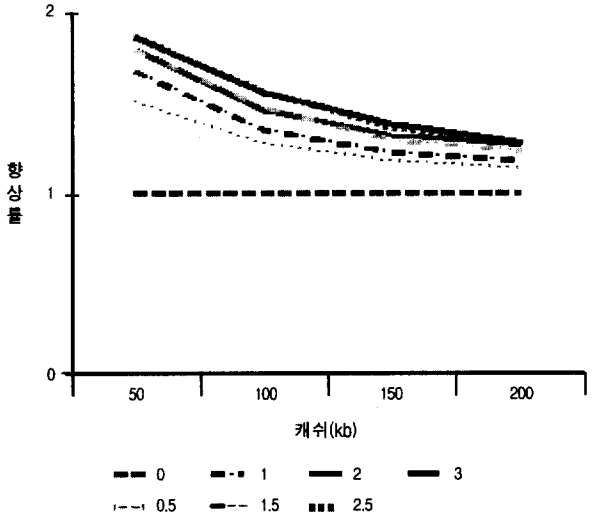
아파치는 사용자의 요청이 들어오면, 문서를 열고, 데이터를 사용자 공간으로 매핑하기 위해 mmap 함수를 호출한다. 그러면, 서버는 read/write 시스템 콜을 피하면서 맵된 데이터를 네트워크로 직접 보낼 수 있다. 전체 파일이 모두 보내지면 파일을 munmap시키고 닫는다. 이것은 시스템 내에서 열 수 있는 파일의 수가 제한되기 때문이다. 웹 문서의 인기도는 Zipf 분포를 따르므로 인기있는 문서는 매우 빈번하게 요청된다. 그러므로 mmap과 munmap은 반복적으로 발생하게된다. 이런 상황에서 파일 데이터가 이미 커널 버퍼에 캐쉬되어 있더라도 버추얼 메모리 시스템은 페이지 폴트를 일으킨다. 왜냐하면 데이터를 모두 보내고 난 후에 매팅된 영역들을 munmap시키고 파일을 닫았기 때문에 파일을 사용자 공간으로 다시 re-mapping시킬 때에 페이지 폴트가 날 것이다. mmap 동작의 사용은 시스템 호출의 오버헤드없이 매핑된 데이터를 직접 네트워크로 보낼 수 있으므로 유저에게 투명성을 제공하지만 페이지 폴트의 오버헤드를 가지므로 mmap으로 매핑된 영역은 시간적 지역성에 따라 캐싱을 하는 것이 웹 서버의 성능 향상에 도움을 준다[9]. 그러나, 문서를 캐싱하는 것보다 디스크에서 직접 TCP/IP 네트워크로 직접 보내는 것이 시스템 성능을 크게 향상시킬 것이다. 현재 Microsoft는 Window NT 시스템에서 *TransmitFile*이라는 API를 지원한다.

Crovella[13]는 웹서버가 처리하는 요청들을 SRPT(Shortest Remaining Processing Time first) 스케쥴링으로 웹서버의 성능을 향상시켰다. SRPT 스케쥴링은 작업의 사이즈를 미리 알아야만 하고, 사이즈가 큰 작업이 기아(starvation)를 일으킬 가능성이 많기 때문에 이론적인 스케쥴링 방법으로만 알려졌다. 그러나, 정적 웹 환경에서는 클라이언트가 요구하는 HTTP 요청들의 사이즈를 미리 알 수 있으므로 SRPT 스케쥴링 방식을 적용시킬 수 있다. 또, HTTP 요청 크기의 분포가 Heavy-tailed이기 때문에 긴 요청들에게 크게 영향을 미치지 않고도 size-independent 스케쥴링에 비해 응답 시간을 향상시켰다. 최근 들어 Heavy tailed 분포는 웹 파일 사이즈의 분포뿐만 아니라 컴퓨터 작업량에 전반적으로 나타나고 있는 현상이다. Heavy tailed 분포는 대다수 파일의 사이즈는 매우 작고, 전체 작업량의 반 이상이 가장 큰 작업의 1% 미만에 해당되는 성격을 갖는다. 그 외 스케쥴링 방식으로 특정 클라이언트 또는 웹 서버내의 특정 문서들에게 우선순위를 두어 요청 품질을 제한하는 애플리케이션 레벨에서

QoS가 있다.

3. 예비실험

대부분의 웹 서버는 동시에 수행해야 할 요청들의 수행 순서를 FCFS (First Come First Service)으로 수행한다. 각 클라이언트들이 서버에 요구 1, 요구 2, 요구 3의 순서대로 문서를 요청하는 경우 서비스 응답도 동일한 순서인 요구 1에 대한 응답, 요구 2에 대한 응답, 요구 3에 대한 응답으로 이루어진다. 그러나 본 논문에서는 디스크 액세스를 줄이기 위해 서비스의 순서를 바꾸는 시뮬레이션을 해보았다. 이 실험은 캐쉬된 문서를 우선 서비스하는 것이 히트율과 바이트 히트율을 향상시킬 수 있는지에 대한 시뮬레이션이며 [18]에서 좀더 구체적으로 서술되어 있다.

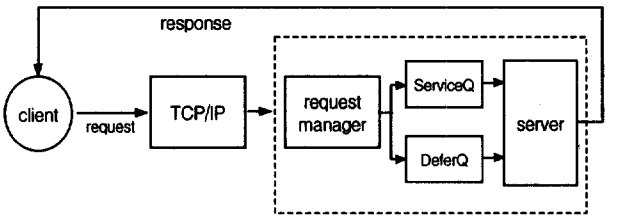


(그림 1) 캐쉬 크기에 따른 서비스 양보의 성능 향상률

실험은 Pentium 350 Mhz, Linux kernel 2.2.16 운영체제를 사용하였다. 서비스 기다림은 0초부터 3초까지 0.25초 간격으로 실험하였다. 서비스 기다림이 0초 일 때는 기다리지 않는 것을 의미하므로 서비스 기다림을 적용하지 않은 것을 나타내는 것이다. 시간 간격을 크게 잡은 이유는 버클리 버클이 대학의 전자계산학과 실제 로그(2000년 7월의 로그 중 1일부터 7일까지의 아파치 웹 서버 로그[14])가 1초에 5~6개의 요구 밖에는 일어나지 않았기 때문이다. (그림 1)은 캐쉬 크기에 따른 서비스 양보의 성능 향상율을 보여준다. 캐쉬되지 않은 문서의 서비스를 기다려 줌으로써 처음 1초에는 많은 향상이 있다.

4. 더블 큐 웹 서버

본 논문에서는 Event_Driven 방식의 웹 서버을 구현하여 더블 큐를 적용하였다. Event_Driven 방식의 웹 서버는 소켓에서 non-blocking read/write, accept 시스템 호출을 사용하며 event completion을 체크하기 위해 select 시스템 호출이 사용된다. 자주 요청되는 문서들은 웹 서버의 사용자 영역에 mmap으로 매핑된 영역을 캐쉬한다. 다음의 (그림 2)은 본 논문에서 제시하는 웹 서버의 전체적인 구성도 이다. 웹 서버는 요청관리자(request manager), 기다림 큐(DeferQ), 서비스 큐(ServiceQ), 서버(server)로 구성된다.



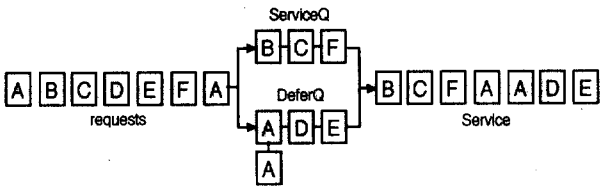
(그림 2) 더블큐 서버 구조

사용자의 요청이 웹 서버의 TCP/IP 리슨큐(Listen Queue)에 도착하여 accept( )되면 요청 관리자는 각 컨넥션에서 일어날 수 있는 이벤트를 탐지할 수 있도록 컨넥션 ID를 부여하고 이벤트 비트맵을 초기화시킨다. 스케쥴링 윈도우내[1]의 요청들 중 read/write 이벤트가 발생하면 요청 관리자는 해당 문서가 캐쉬되어 있는 지를 확인한다. 만약 캐쉬되었다면 즉시 서비스될 수 있도록 서비스 큐로 넘겨주고, 그렇지 않다면 기다림 큐로 이동시킨다. 서버는 스케쥴링 윈도우 내의 요청들 중 캐쉬된 문서들을 모두 서비스한 후 캐쉬되지 않은 문서들의 요청을 서비스한다. 스케쥴링 윈도우의 크기는 select 또는 poll 함수가 수행되는 주기가 된다.

4.1 캐쉬 교체 정책의 모듈성

문서의 요구 순서가 A, B, C, D, E, F, A이라고 하자(그림 3). 그리고 캐쉬에는 B, C, F 문서만이 캐쉬되어 있다고 가정하자. 문서 F, C, B는 서비스 큐로 들어가자마자 서비스 받는다. 캐쉬에 없는 문서인 A, D, E는 기다림 큐로 들어가게 되고, 서비스 큐의 문서들에 대한 서비스가 완료되면 기다림 큐의 문서들이 서비스된다. 이때 기다림 큐의 같은 문서인 A는 연속적으로 서비스된다. 결국 서비스 순서는 B, C, F, A, A, D, E로 변경된다. 서비스 기다림이 끝난 후에 미스가 일어나는 문서들의 서비스로 인해 시간적 참조 지역성이 파괴되었음을 알 수 있다. 따라서 본 논문에서는 서비스 기다림은 적용하지만, 바뀐 서비스 순서가 캐쉬 교체 정책에는 영향을 미치지 않도록 한다. 즉, 웹 서버가 어떤 캐쉬 교체 정책을 이용하더라도 교체 정책을 방해하지 않는 것을 의미한다. 그러므로, 본 논문에서의 더블 큐 서비스 정책은 특정 캐쉬 정책에 제한되지 않음으로써 모듈성을 제공한다.

1) 본 논문에서는 동시에 수행해야 할 수십 내지 수 백 개의 요청들을 스케쥴링 윈도우라 칭한다.



(그림 3) 더블 큐 웹 서버의 서비스

4.2 semi-LRU 메인 메모리 캐싱

파일 시스템과 데이터 베이스에서 연구 개발되었던 상당 수의 캐쉬 교체 정책들은 웹 캐싱에 잘 적용되지 않는다. 웹 서버에서는 문서 전체를 캐쉬해야 하므로 문서마다 캐싱의 단위가 같지 않기 때문이다. 본 논문에서의 더블 큐 웹 서버는 LRU 교체 기법과 비슷한 *semi*-LRU 교체 방식을 채택하였다. *semi* -LRU는 문서들을 참조된 순서가 아니라 요청된 순서대로 캐쉬 리스트를 유지하므로 시간적 지역성을 유지한다. 다음의 <표 1>은 LRU와 *semi*-LRU의 차이점이다.

〈표 1〉

	LRU	*semi*-LRU
캐싱 단위	블럭	문서
교체 대상	가장 오래 전에 참조된 블럭	가장 오래 전에 요청한 문서

본 논문에서 *semi*-LRU 교체 방식을 선택한 것은 가장 성능이 우수하기 때문은 아니다. 비교적 구현이 쉽고 간단하기 때문이다. 버블 큐 웹 서버는 캐쉬 교체 정책에 대한 모듈성이 있으므로 어떤 방식의 교체 방식도 대체될 수 있다.

4.3 서비스 기다림

예비 실험에서는 캐쉬되지 않은 문서의 서비스를 인위적으로 지연시켰다. 캐쉬된 문서를 먼저 서비스하는 것이 히트율을 향상시키므로써 웹 서버의 성능 향상에 도움을 주는지에 대한 실험이었기 때문이다. 그러나, 더블 큐를 지원하는 웹 서버에서는 기다림 큐의 요청들은 서비스 큐의 모든 캐쉬된 문서들이 수행을 마칠 때까지만 기다리도록 하였다. 초당 수십내지 수 백개의 요청들이 도착하는 바쁜 웹서버에서 인위적인 지연 시간을 설정하기에는 다소 무리가 있다. 그러므로, 캐쉬되지 않은 문서의 기다림 시간은 캐쉬 된 문서들의 서비스 시간과 먼저 선택된 캐쉬되지 않은 문서들의 서비스 시간의 합이 된다. 스케쥴링 윈도우내의 요청의 수는 $N(n+m)$, 서비스 큐의 길이는 n, 기다림 큐의 길이는 m이라고 하자. 캐쉬되지 않은 문서의 최소 기다림 시간(*min_Defer_Time*)은 $\sum_{i=1}^{n} serviceQ_i$이고, 최대 기다림 시간(*max_Defer_Time*)은 *Min_Defer_Time*과 $\sum_{j=1}^{m-1} DeferQ_j$의 합이다.

5. 실험 데이터

본 실험에서 사용한 웹 서버에 존재하는 정적 문서는 2000개이며 이들의 성격은 <표 2>과 같다. 서버에 존재하는 문서의 크기 분포와 요청되는 문서의 크기 분포는 바디와 테일로 나누게 되는데 바디(body) 부분은 로그노르멀 분포를 따르며 테일(tail) 부분은 빠레또 분포를 따른다. 이들 각 분포의 바디와 테일의 구분점은 93% 지점이 된다. 요청되는 문서의 빈도 수는 zipf 분포를 따른다. <표 2>의 수치는 실제 웹 데이터의 분석을 통해 얻어진 수치이며[8] 이것을 SURGE 클라이언트 생성기에 적용하여 실험하였다. SURGE(Scalable URL GEnetator)[8]는 문서의 인기도, 문서의 크기, 요청 크기, 시간적 지역성, 공간적 지역성, 임베디드된 문서의 수, 오프타임에 대한 분포가 웹 성격을 잘 나타내지도록 정적URL 스트림을 자동 생성 한다. 즉, 생성되는 요청들은 웹상의 실제 데이터 성격을 잘 표현하는 스트림이다.

〈표 2〉

SURGE Client	
서버에 존재하는 문서의 크기 분포	Logormal (μ = 7.630, α = 1.001)
	Pareto (k = 10000, α = 1.0)
	Cutoff 0.93
요청되는 문서의 크기 분포	Logormal (μ = 7.881, α = 1.339)
	Pareto (k = 34102, α = 1.177)
	Cutoff 0.93
요청되는 문서의 인기도	$\Omega \frac{1}{i^{\alpha}}$ (Ω *constant* $\alpha = 1$, i : *ranking*)

6. 성능 평가 실험

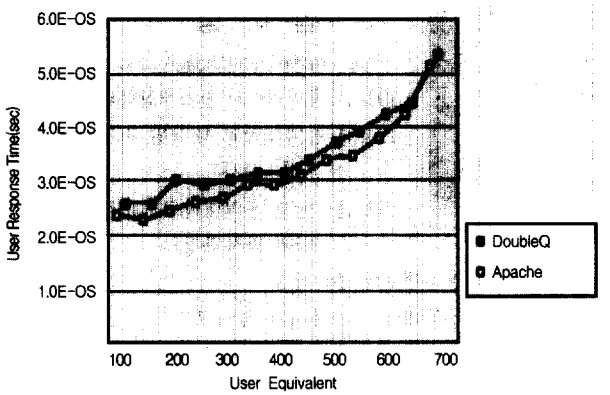
본 논문에서 사용된 실험장비는 다음의 <표 2>와 같다. 클라이언트 머신은 100개~700개의 스레드(UE : User Equivalent)가 HTTP 요청들을 연속적으로 생성해 내어 서버가 고 부하 상태가 되도록 한다. 실험 결과는 실험실 내의 가까운 거리에서 100Mbps Ethernet환경에서 실험한 것이다. 만약, 사용자와 서버가 지역적으로 먼 거리에 있는 WAN 환경에서는 평균 사용자 응답 시간이 비교적 길 것이다.

〈표 3〉 실험장비

	RAM	CPU	OS
클라이언트	256M	Pentium IV 1GHz	LINUX 2.4
서 버	128M	Pentium II 350MHz	LINUX2.2.16

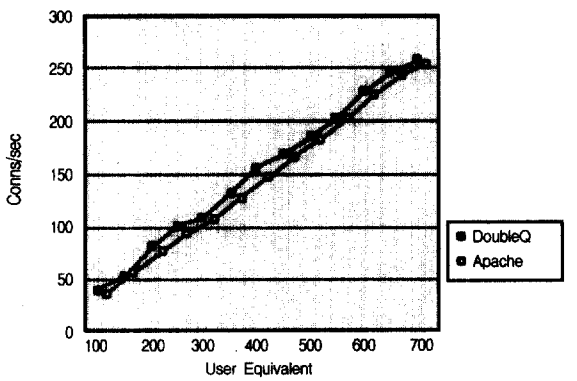
웹서버의 성능은 사용자 응답 시간(Service Response Time)과 처리율(Throughput)로 웹 서버의 성능을 평가 할 수 있다. 사용자 응답 시간은 클라이언트에서 측정할 수 있는 웹 서버의 평가 요소이며 처리율은 웹 서버에서 측정할 수 있다.

평균 사용자 응답 시간이란 클라이언트의 요구가 서버까지 도착하는데 걸리는 시간($t1$)과 웹 서버에서 서비스하는데 걸리는 시간($t2$)과 웹 서버의 응답이 다시 클라이언트까지 도달하는데 걸리는 시간($t3$)의 합을 모든 요청의 수(ri)로 나눈 값을 말한다. 처리율은 웹 서버에서 초당 처리되는 요청의 수(컨넥션 비율)이다. 본 논문에서는 가장 대중적인 웹 서버인 아파치(Ver1.3.9)와 성능을 비교해 보았다. 아파치 웹 서버의 환경 설정은 기본 설정을 그대로 유지하고, 본 논문에서의 더블 큐 웹 서버도 아파치와 동일한 환경을 갖도록 설정하였다.



(그림 4) 사용자 응답시간

(그림 4)는 사용자 응답 시간의 비교이다. 비교적 고부하인 100UE부터 700UE까지의 사용자 응답 시간을 비교해 볼 때 더블 큐를 지원하는 본 논문에서의 웹 서버가 응답시간을 평균 5%이상 향상시킨다.



(그림 5) 웹서버의 처리율

(그림 5)는 웹 서버에서 측정한 처리율이다. 본 논문에서의 더블 큐 웹서버는 아파치 웹 서버의 처리율을 약간 상회한다.

7. 결론 및 향후 과제

정적 웹 환경에서 동시에 수 백 내지 수천 개의 요청을 서비스해야 하는 웹서버의 오버헤드는 디스크 접근이다. 디스크 접근 오버해드를 줄이기 위해 대부분의 웹 서버들은 메인 메모리 캐싱을 하지만 들어오는 요청들의 순서를 제어하지는 않는다. 본 논문에서 제시한 웹 서버는 디스크 접근 빈도 수를 줄이기 위해 동시에 서비스해야 할 요청들을 두 가지로 분류한다. 캐쉬되어 있는 문서를 요구하는 요청은 먼저 서비스하며 캐쉬되지 않은 문서를 요구하는 요청은 기다림 큐에 넣는다. 웹 서버는 캐쉬된 문서를 요구하는 요청들을 모두 서비스 한 후 기다림 큐의 요청을 서비스한다. 본 논문에서의 더블 큐 웹 서버는 서비스 순서를 바꾸지만 웹 캐싱 정책을 방해하지 않기 때문에 모듈성을 갖는다. 그러므로, 기존의 대부분의 웹 캐싱 정책을 적용할 수 있다. 본 논문에서는 더블 큐 웹 서버의 동작을 평가하기 위해 실제 웹 데이터의 성격을 갖는 웹 클라이언트 자동 생성기[8] 이용하여 실험하였다. 디스크 접근 오버헤드를 줄이기 위해 캐쉬된 문서의 서비스를 우선하는 정책으로 웹 서버의 처리율과 사용자 응답 시간을 향상시켰다.

사용자의 요청이 많은 고부하에서는 성능이 우수했지만, 사용자의 요청이 상대적으로 적은 고부하에서는 비슷한 모습을 보였다. 그러므로, 부하가 적을 때는 본 논문에서의 더블 큐 방식에 대한 큰 잇점이 없으므로, 부하의 정도에 따라 더블큐 서비스를 적응하는 연구가 필요하다.

참 고 문 헌

[1] "The Internet, Technology 1999, Analysis and Forecast," IEEE Spectrum, January, 1999.

[2] Nina Bhatti and Rich Friedrich, "Web server support for tiered services," In IEEE Network, Hui Zhang and Edward W. Knightly, Eds. IEEE Communications Society, September, 1999.

[3] N. Bhatti, A. Bouch, and A. Kuchinsky, "Integrating User-Perceived Quality into Web Server Design," In 9th International World Wide Web Conference (WWW9), Amsterdam, May, 2000.

[4] J. Almeida, and P. Cao, "Measuring Proxying Performance with the Wisconsin Proxy Benchmark," In Journal of Computer Networks and ISDN Systems, 1998.

[5] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, "Ex-ploring the Bounds of Web Latency Reduction from Caching and Prefetching," In Proc. of the USENIX Sym-posiumon Internet Technologies and Systems, 1997.

[6] Vigilio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira, "Characterizing Reference Locality in the WWW," In IEEE International Conference in Parallel and Distributed Information Systems, Miami Beach, Florida, USA, December, 1996.

[7] Evangelos P.Markatos. "Main Memory Caching of Web Documents," In Fifth Internation World Wide Web Conference May 6~10, 1996.

[8] Paul Barford and Mark Crovella, "Generating Representative Web Workloads for Network and Server Performance Evaluation," In Proceedings of SIGMETRICS '98, pp.151-160, July, 1998.
[9] V. S. Pai, P. Druschel, and W. Zwaenepoel, "Flash : An efficient and portable Web server," In Proceedings of the USENIX 1999 Annual Technical Conference, pp.199-212, June, 1999.
[10] Apache. http : //www.apache.org.
[11] Zeus Technology Limited. http : //www.zeus.co.uk.
[12] http : //squid-cached.org.
[13] Mark E. Crovella, Robert Frangioso, and Mor Harchol-Balter, "Connection Scheduling in Web Servers," In 2nd USENIX Symposium on internet Technologies and Systems, 1999.
[14] http : //www.cs.berkeley.edu/logs/.
[15] Usability Engineering. Academic Press, 1993.
[16] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel and Erich Nahum, "Locality-Aware Request Distribution in Cluster-based Network Servers," In Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII), San Jose, CA, October, 1998.
[17] Mohit Aron, Peter Druschel and Willy Zwaenepoel, "Efficient Support for P-HTTP in Cluster-Based Web Servers," In Proceedings of the USENIX 1999 Annual Technical Conference, Monterey, CA, June, 1999.
[18] 장혜진 "웹 서버 성능 향상을 위한 서비스 기다림", 홍익대학교 대학원 석사학위 논문, 2001.

염 미 령

e-mail : miryeom@cs.hongik.ac.kr
1992년 홍익대학교 전자계산학과 졸업(학사)
1994년 홍익대학교 대학원 전자계산학과 졸업 (석사)
1999년~현재 홍익대학교 대학원 전자계산학과(박사과정)
1997년~현재 현대서포트(주) 과장
1999년~현재 안양대학교 컴퓨터과 겸임교수
관심분야 : 웹 서버, 웹 서버 클러스터링, Web QoS