2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

동적 메인 메모리 웹 캐쉬 할당 정책

염미령
홍익대학교 전자계산학과
e-mail:miryeom@cs.hongik.ac.kr

Dynamic Main Memory Web Caching

Mi-ryeong Yeom
Dept. of Computer Science, Hong-Ik University

요약

웹의 이용율 증가로 클라이언트의 요청이 급증하지만 웹 서버의 처리 능력은 한계에 다다르고 있다. 늘어나는 요청율에 응답하기 위해서 웹 서버는 불필요한 오버해드는 피해야 한다. 본 논문에서는 요청들에 대한 처리 순서를 고려함으로써 입출력 오버헤드를 줄이는 동적 메인 메모리 캐싱을 제안하였으며 실험 결과 웹 서버의 처리율을 증가시켰다.

1. 서 론

웹 사용이 점점 증가함에 따라 웹 서버의 처리 능력은 한계에 다다르고 있다. 그러므로, 늘어나는 요청율에 응답하기 위해서 웹서버는 불필요한 오버해드는 피해야 한다. 웹 서버의 오버해드는 디스크 액색스이다. 클라이언트가 웹 서버에게 문서를 요청하면, 웹 서버는 파일 시스템을 호출하여 파일을 open/close한다. 웹 서버는 파일 시스템 캐쉬에 의존하기 때문에 이들 시스템 호출은 하나 또는 그 이상의 디스크 액색스를 하게된다. 만약에 문서가 서버의 로컬 디스크에 존재하지 않는다면 네트워크 파일 시스템으로부터 가져와야 하는 아주 큰 오버해드가 발생할 수 있다.

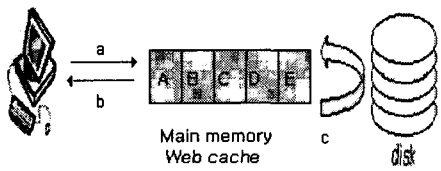

그림 1로 예를 들어보면, 사용자가 요구(a)하는 문서가 캐쉬에 있다면 곧바로 서비스(b) 되지만 캐쉬에 없다면 디스크에서 가져오는 (c)시간 동안 응답은 지연된다. 웹 서버의 응답이 지연되면, 사용자는 더 이상 기다리지 못하고 컨넥션을 끊은 후 다시 같은 요청하게 된다. 이런 행위는 TCP 컨넥션을 열고 닫는데 많은 비용을 들이므로 바쁜 서버에게 부하를 더욱 더 가중시키는 결과를 낳는다.

최근 들어 웹 클라이언트의 요구에 대한 수요 예측을 못할 정도로 인터넷 사용은 급증하여 서버의 응답시간은 길어지고 사용자의 인내심은 짧아졌다[10]. 그러므로, 서버는 불필요한 오버해드를 최소화하여 과적 상태일 때에도 사용자가 납득할 수 있는 처리율과 응답시간을 보장해 주어야 한다.

파일 시스템 캐쉬는 서버의 메인 메모리에 가장 인기 있는 파일 블록들을 캐쉬함으로써 디스크 액색스의 수를 줄일 수 있다. 그러나, 전통적인 파일 시스템의 캐쉬는 다음과 같은 이유로 웹 트래픽에 잘 적응하지 못한다. 첫째,전통적인 파일 시스템에서는 블록을 캐쉬하지만 웹 캐쉬는 파일 전체가 캐쉬되야 하므로 캐싱의 단위가 틀리다. 웹 캐쉬는 디스크 액색스를 한번만으로도 파일 전체를 가져오도록 문서를 요청할 수 있지만 파일 시스템에서는 한번에 하나 혹은 몇 개의 블록만을 가져온다. 둘째, 요청한 문서가 파일 캐쉬 내에 있다하더라도 웹서버는 그 파일을 열거나 닫기 위해 파일 시스템 호출을 해야 하는 오버해드가 존재한다. 셋째, 대부분의 웹 요청은 문서의 내용을 바꾸지 않는 읽기 전용 접근이다. 그러나 파일 시스템에서는 읽기 전용 또는 읽기/쓰기 모두를 처리 할 수 있다. 읽기 전용 접근은 읽기/쓰기 접근에 비해 데이터 블록을 여러 번 카피해야하는 오버해드를 피할 수 있다.

고성능 웹 서버는 동시에 수십 내지 수백 개 이상의 요청들을 처리할 수 있으며 이들의 효율적인 처리를 위해 메인 메모리 캐싱을 한다. 그러나, 대부분의 웹 서버는 동시에 수행해야 할 요청들의 수행 순서를 운영체제에 의존하며 웹 서버 내에서 특별한 스케쥴링을 하지는 않는다. 그러나 만약, 클라이언트가 요청하는 문서에 대한 정보를 미리 알 수만 있다면, 동시에 들어오는 요청들에 우선 순위를 두어 수행시킬 수 있으므로 운영체제 의존적인 기존의 처리 방식보다는 웹 서버의 성능을 높일 수 있다.

본 논문에서는, 동시에 들어오는 수십 내지 수 백개의 요청들을 스케쥴링 윈도우라 칭한다. 스케쥴링 윈도우내의 요청들이 요청하는 문서들 중 가까운 미래에 참조될 문서에 가치를 두어, 이들이 메인 메모리 캐쉬 내에 더 오래 머물게 하는 동적 메인 메모리 캐쉬 할당 정책 제안한다. 실험 결과, 동적 메인 메모리 캐쉬 할당 방식을 적용한 웹서버는 요청들의 실행 순서를 고려치 않은 웹 서버보다 초당 요청 처리율이 최대 15%이상 우수함을 보였다.

본 논문에서는 가장 성능이 우수한 것으로 알려진 AMPED 방식의 FLASH[1] 웹 서버의 모듈을 수정하였으며 클라이언트는 정적 요청들을 자동으로 생성해주는 SURGE[6]를 구동시켰다. 요청들은 HTTP 1.0 GET 메소드들이며 정적 문서를 요구한다. 문서의 인기도(popularity)는 zipf[3] 분포를 따르며 서버에 존재하는 파일의 분포는 heavy-tailed[3] 분포이다.

본 논문의 제2절에서는 동적 메인 메모리 내캐쉬 할당 정책을 제시하고, 제3절에서는 관련 연구에 대해 알아보며, 제4절에서는 실험 및 결과를 그리고 제5절에서는 결론 및 향후 과제를 끝으로 논문을 맺는다.

2. 동적 메인 메모리 캐쉬 할당 정책

동적 메인 메모리 캐쉬 할당 정책은 스케쥴링 윈도우내의 요청들의 수행 순서를 메인 메모리 캐싱에 반영한다. 이것은 가까운 미래에 참조될 문서들의 목록을 파악하여 문서들이 참조되는 시점까지 캐쉬 내에 머물게 하려는 의도이다. 스케쥴링 윈도우 내에서 각 요청들이 요구하는 문서가 메인 메모리에 캐싱 되었는지를 탐색한다. 만약 캐쉬 되었다면 그 문서는 캐쉬에서 가치 있는 문서로 재조정된다. 즉 가장 가까운 미래에 사용될 문서이므로 메모리 버퍼에서 교체가 발생할 때 가장 나중에 쫓겨나는 문서가 되는 것이다. 스케쥴링 윈도우내의 모든 요청들에 대한 캐쉬 조정이 모두 끝나면, 웹 서버는 스케쥴링 윈도우 내의 요청들을 수행시킨다. 이때, 캐쉬되어 있는 문서를 요구하는 요청들과 캐쉬되지 않은 문서를 요구하는 요청들은 우선 순위의 구분 없이 MP방식으로 수행된다.

동적 메인 메모리 캐쉬 할당 정책은 스케쥴링 윈도우 내에 존재하여 가까운 미래에 참조될 문서들이 캐쉬 미스(cache miss)되는 확률을 줄이기 위해 캐쉬 내의 문서들 가치를 조정하는 것이다.

이 방법은 캐쉬된 문서를 요구하는 요청들과 캐쉬되지 않은 문서를 요구하는 요청들이 혼합되어 수행될 지라도, 가까운 미래에 참조될 예정이며 캐쉬되어 있는 문서의 가치를 높이 평가하여 쫓겨날 확률을 줄이는 것이다.

2.1 동적 메인 메모리 캐쉬 할당정책의 구현

스케쥴링 윈도우의 비트맵을 스캔하면서 요구하는 문서가 캐쉬에 있는 지를 확인한다. 만약 캐쉬에 있다면 그 문서는 캐쉬 리스트의 맨 앞로 이동시킨다. 이런 방식으로 스케쥴링 윈도우의 모든 요청들에 대한 캐쉬 리스트의 조정이 모두 끝나면, 캐쉬 리스트는 새로이 구성된다. 가까운 미래에 참조될 캐쉬내의 문서들은 캐쉬 리스트의 앞부분을 차지하게 된다. 캐쉬 리스트의 조정이 끝난 후, 비트맵에 셋팅된 요청들을 서비스 한다.

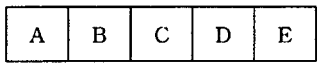

그림2. 동적 메인 메모리 캐쉬 할당 전

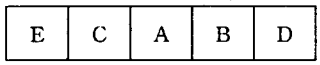

그림3. 동적 메인 메모리 캐쉬 할당 후

예를 들어, 그림 2에서처럼 메인 메모리 캐쉬에 문서 A,B,C,D,E가 캐쉬되어 있고 (A: most recent used. B:least recent used) 캐쉬가 꽉차면 LRU 교체 정책을 따른다고 가정하자. 이때 스케쥴링 윈도우에 셋팅된 문서가 F,C,G,E이고, 특별한 스케쥴링을 수행하지 않는다면 수행 순서는 F-C-G-E가 될 것이고 3번의 캐쉬 미스를 초래할 것이다. 그러나, C와 E가 캐쉬되었다는 정보를 미리 알고 이들을 MRU로 옮겨 캐쉬의 상태가 그림 3에서 처럼 E,C,A.B,D로 조정된 후에 요청 F-C-G-E의 순서대로 수행한다면 2번의 캐쉬 미스가 발생한다.

3.관련연구

Crovella[2]는 웹서버가 처리하는 요청들을 SRPT (Shortest Remaining Processing Time first) 스케쥴링으로 웹 서버의 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 요청들의 분포가 heavy-tail이기 때문에 긴 요청들에게 크게 영향을 미치지 않고도 size-independent 스케쥴링에 비해 응답 시간을 향상시켰다.

[4]에서는 같은 문서를 요구하는 요청들을 연속적으로 수행시켜 응답시간을 향상시켰다.[5]에서는 캐쉬된 문서를 우선 서비스하여 웹서버의 처리율을 높였다. TCP/IP나 HTTP 프로토콜은 우선 순위의 개념이 없으므로 애플리케이션 QoS를 통해 컨넥션을 스케쥴링한다[10]. 성능이 우수한 프락시 웹 캐쉬 교체 알고리즘으로는 큰 문서를 우선으로 쫓아내는 SIZE[7], Greedy Dual 알고리즘을 적용한 GD-SIZE[8]등이 있다. 이들을 메인 메모리 웹캐쉬 교체 정책으로 이용 할 수 있으며, [9]에서는 512 Kbytes의 메모리를 캐쉬로 이용하여도 전체 요청 문서의 60% 이상을 저장하기에 충분함을 보여주었다. [1]에서도 실험을 통해 메인 메모리 캐싱이 웹 서버의 성능을 향상시킴을 보여주었다. 클라이언트들은 뒤 (BACK), 앞 (FORWARD) 단추를 이용해 짧은 시간 내에 같은 문서를 반복 접근한다[9]. 그리고 문서들은 재참조될 확률이 zipf[3] 분포를 따르므로 자주 참조되는 문서들을 메인 메모리에 캐쉬한다면 재 접근시에 히트되어 클라이언트의 응답 시간도 줄일 수 있다. 또한 웹 서버의 부하도 상당량 줄일 수 있게 된다.

4.실험 및 결과

4.1 실험 환경

우리의 실험는 웹서버와 클라이언트를 다음과 같이 셋팅하였다. 웹서버는 LINUX 2.2.16, 128M RAM, 펜티엄 II 350 Mhz에서 구동되며 클라이언트는 128M RAM, Pentium III 750 Mhz에서 수행된다. 웹 문서는 시간적 지역성을 나타내며 요구된 문서의 빈도 수는 zipf[3] 분포를 따른다. 서버에 존재하는 파일은 총 2000개이며 바디는 로그노르말(lognormal)이며 꼬리(tail)은 heavy-taild[3] 분포이다(그림 4).

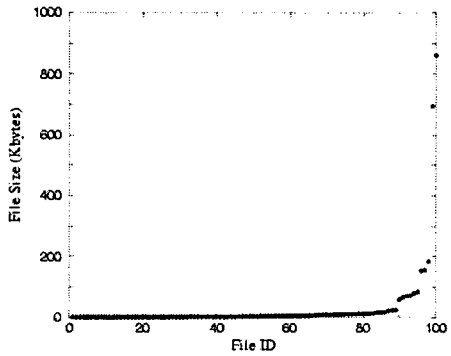

그림 4 문서의 분포

4.2 클라이언트와 요청되는 문서

웹 문서는 시간적 지역성을 나타내며 요구된 문서의 빈도 수는 zipf[3] 분포를 따른다[그림 5). 본 논문에서는 이러한 특징을 갖는 요청을 자동으로 생성해 주는 생성해주는 SURGE[6]를 구동시켜 실제의 웹 환경과 유사한 모델로 실험하였다. 클라이언트 머신은 250개~400개의 스래드(UE: User Equivalent)가 HTTP 요청들을 연속적으로 생성해 내어 서버가 고부하 상태가 되도록 한다.

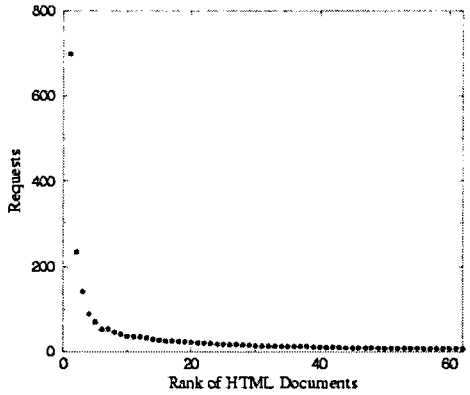

그림 5 문서의 인기도

4.3 성능평가

4.3.1 사용자 응답시간

동적 메인 메모리 캐쉬 할당 정책과 LRU를 비교하였다. 400 UE 미만일 때는 동적 메인 메모리 캐쉬 할당 정책이 우수하지만, 400UE 일 때는 0.1초 더 걸렸다. 그러나, 이 정도의 시간차는 사용자가 감지 할 수 없으므로 큰 의미는 없다. 사용자 응답 시간이란 클라이언트의 요구가 서버까지 도착하는데 걸리는 시간($t1$)과 웹 서버에서 서비스 하는데 걸리는 시간($t2$)과 웹 서버의 응답이 다시 클라이언트까지 도달하는데 걸리는 시간($t3$)의 합을 의미한다. 평균 사용자 응답 시간(Average Service Response Time)은 다음의 수식으로 표현될 수 있으며 그림 6에서는 동적 메인 메모리 캐쉬 할당 정책을 적용한 웹 서버와 적용치 않은 웹서버를 비교하였다.

$$ASRT = \sum(t1 + t2 + t3)/\sum ri$$

4.3.2 웹서버 처리율

처리율은 웹 서버에서 측정될 수 있는 메트릭으로 성능 평가의 기준이 된다. 본 논문에서는 초당 처리되는 요청의 수를 표현한다. 그림 7에서는 웹서버에서의 성능을 평가하기 위해 초당 수행되는 요청들의 수를 비교하였다. 동적 메인 메모리 캐쉬 할당 정책을 적용한 웹 서버가 요청들의 실행 순서를 고려치 않은 웹 서버보다 초당 요청 처리율이 항상 우수함을 보인다.

5. 결론 및 향후 과제

우리는 동시에 들어오는 요청들에 대한 정보를 미리 알아내어, 이들의 순서를 캐쉬에 반영함으로써 웹 서버의 처리율이 향상됨을 실험을 통해 보여주었다. 본 논문에서는 가까운 미래에 참조되는 문서를 미리 파악하여, 메인 메모리 캐쉬에서 가장 가치가 높은 문서로 승격시킨 후 요청들을 수행하는 동적

메인 메모리 캐쉬 할당 정책을 제시하여 성능이 향상됨을 실험을 통해 보여주었다.

본 논문에서는 정적 데이터만을 고려한 단일 웹서버 환경에서 실험하였지만, 확장성과 이용성을 극대화하기 위해 웹서버의 노드를 분산시키는 웹서버 클러스터링 환경으로 확장된 스케쥴링이 요구된다. 뿐만 아니라 최근 들어 점점 증가하는 동적 데이터들에 대한 연구도 요구된다.

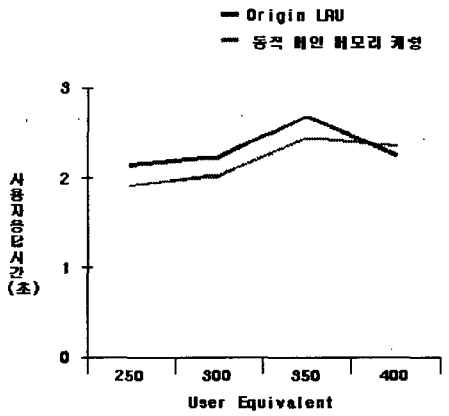

그림 6 사용자 응답 시간

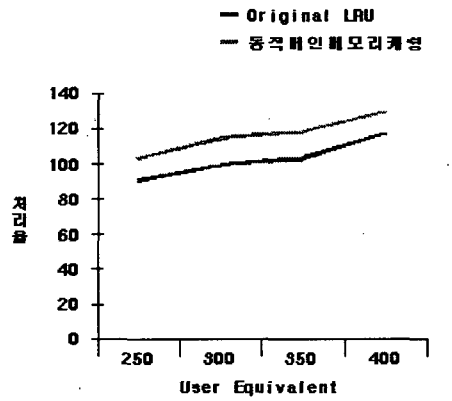

그림 7 웹 서버의 처리율

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] 염미령,노삼혁. "우선 순위를 갖는 웹서버 컨넥션 스케쥴링". 한국정보과학회 학술 발표 논문집(III) (제27권2호)

[5] 염미령,노삼혁. "Flash 웹서버에서 캐쉬된 문서의 우선 서비스". 한국정보과학회 학술 발표 논문집 (A) (제28권1호)

[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] S.Williams, M.Abrams, C.Standridge, G.Abdulla, and E. Fox, "Removal policies in network caches for world-wide web documents," In Proceedings of the ACM SIGCOMM '96, pp.293-305,1996

[8] P.Cao and S. Irani, "Cost-Aware WWW Proxy Caching Algorithms", In Proceedings of the USENIX Symposium on Internet Technologies and Systems. pp.193-206,1997

[9] Evangelos P.Markatos, "Main Memory Caching of Web Documents", In Fifth Internation World Wide Web Conference May6-10, 1996, Paris, France.

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