Coloured Petri-Net을 이용한 커널 스레드 웹 가속기 모델링

황준*, 민병조*, 김학배*
*연세대학교 전기전자공학과
e-mail:hbkim@yonsei.ac.kr

A Modelling of Kernel-Thread Web Accelerator Using Coloured Petri-Net

June Hwang*, Byung-Jo Min*, Hag-bae Kim*
*Dept. of Eletrical and Eletronic Engineering, Yonsei University

요 약

인터넷 인구의 폭주로 트래픽을 줄이려는 노력이 일고 있는 가운데, 우리는 기존에 웹 서버의 성능을 리눅스 커널 단에서 향상시키는 방법을 제안하였다. 사용자 영역의 스레드를 사용하지 않고 커널 영역의 스레드를 사용하고 CPU 개수와 동일한 스레드만 사용함으로서 스레드간의 컨텍스트 스위칭과 시스템 콜 사용의 오버헤드를 줄일수 있었다. 이번에는 이 구조를 CPN(Coloured Petri-Net)을 사용하여 논리적으로 분해하고 그 동작특성을 이해하여 모델링과 실제 데이터를 비교함으로서 프로그램 동작에 관한 논리적 문제점을 발견할 수 있고, 동작특성의 시간특성을 알 수 있다.

1. 서론

이미 서버의 성능향상에 대한 많은 연구들이 있어왔다. 그러나 대다수의 방법들이 부하 분산을 위한 별도의 독립형 서버 장비의 추가를 원하거나 서버 앞 단에 별도의 캐싱 서버를 두는 방법, 또는 응용프로그램 내에서 프로토콜이나 메타 데이터와 관련된 데이터의 조작등을 통하여 서버의 성능향상을 꾀하였다. 그러나 이러한 방법들은 별도의 장비의 구입과 관리, 운영이라는 단점을 갖고 있고, 응용 프로그램 내에서의 성능향상이라는 한계 역시 가지고 있다.

이런 단점들에 반하여 본 논문에서는 대다수의 서버에 장착되어 있는 운영체제, 특히 리눅스로 운영되는 서버들은 웹 서비스 수행에만 시스템의 자원을 사용한다는 점에 착안하여, 커널 자체에 웹 서비스를 가속 수행하는 구조를 추가하게 되었다. 이 방법으로 커널이 무거워질 수 있지만 대다수의 웹 서버는 단지 웹 서비스의 목적만으로 사용되고 있으므로 이러한 역기능을 피할 수 있다. 우리는 이러한 단점을 극복할 수 있도록 리눅스 커널 단에서 클라이언트 측에서 요청하는 요청을 선별적으로 미리 처리하게 하였다. 우선 사용자 수준 스레드를 사용하지 않고 커널 수준 스레드를 사용한다. 대다수의 웹 서버 소프트웨어는 사용자 수준 스레드를 사용한다. 해당 소프트웨어 자체가 커널이 아닌 응용 프로그램이기 때문이다. 이 경우 요청 처리 및 스레드 간의 스위칭에 소모되는 시스템 자원이 낭비된다. 또한 클라이언트의 요청을 받아들일때 기존 응용프로그램, 예를 들면 Apache,의 경우 들어오는 요청마다 스레드를 할당하였는데, 우리는 한개의 CPU가 스레드 한개를 처리하는 동안 다른 요청들을 모아 둘수 있는 자료 구조를 제안한다. 이것은 각각의 사용자 수준 스레드에 메모리를 할당하는데 발생하는 메모리 낭비를 줄일 수 있다. 이 외에, 커널단에 추가적으로 캐시 알고리즘을 도입하여 클라이언트 요청에 좀 더 빠르게 반응 할 수 있도록 하였다.

2. 기존 웹서버의 문제점

아파치는 오늘날 가장 많이 쓰이는 웹서버 응용프로그램으로 대부분 리눅스 플랫폼에서 작동한다. 아파치는 boss/worker 멀티 쓰레드 모델을 채용하고 있다. 이것은 thread per request 모델이라고도

하며, boss 쓰레드가 각각의 사용자 요청을 받아들여서 그 요구에 맞는 태스크(task)들을 발생시키고 그 태스크는 worker 쓰레드로 할당된다. Worker 쓰레드는 우선 해당 요청의 객체가 메모리에 캐시 되었는지 확인하고 캐시 되어 있다면 클라이언트에게 보내고 그렇지 않으면 디스크로부터 메모리로 읽어와서 캐시하고 그것을 클라이언트로 보내게 된다. 각각의 요청에 대해 worker쓰레드마다 별도의 독립성을 가지고 요청을 수행한다. 아파치는 사용자 레벨의 응용프로그램이므로 사용자 쓰레드를 사용하게 된다. 사용자 쓰레드를 이용할 때, 시스템 콜을 사용하여 반드시 커널 내로 들어와 커널 함수와 라이브러리를 사용해야 하는데 그 과정에서 스위칭 시간 및 메모리 오버헤드가 발생한다. 따라서, 동시에 많은 사용자 요구가 들어올 경우, 발생되는 사용자 쓰레드가 늘어나게 되고, 그에 따라 각각의 사용자 쓰레드에 할당된 메모리는 증가하게 되고 그 쓰레드간의 스위칭이 빈번하게 일어남으로써 불필요한 시간의 낭비가 발생하게 된다. 그림 1은 아파치 웹 서버의 내부 동작 구조이다. [1]에서는 실제로 user-level의 스레드를 사용하여 클라이언트의 매 요청을 각각 하나의 스레드에 할당하였다. 그리고 그 요청의 빈도를 높여가며 작업처리량(throughput)을 측정한 결과, 특정 요청의 개수가 동시에 스레드에 할당되었을때 까지는 처리량이 증가하였으나, 그 이후로는 급격한 성능감소를 보였다. 상대적으로 편리한 프로그래밍 방법에도 불구하고, threading 과 관련된 오버헤드들 - cache와 TLB 미스, 스레드 사이의 스케줄링 오버헤드, 공유 리소스의 lock 경쟁 등- 이 스레드의 개수가 커질때 마다 성능의 심각한 감쇠를 가져오게 된다.

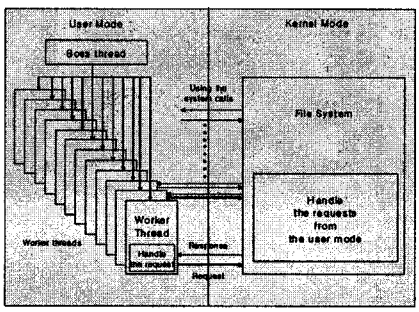

그림1 아파치의 내부 동작구조

3. 개선된 웹 가속기의 구조

이전에 우리가 개발한 가속기의 구조는 평문 요청만을 처리하는 부분과 SSL세션 요청을 처리하는 부분으로 나눌 수 있다. 평문 요청 가속부분과 SSL 요청 가속부분에, 공통적으로 커널영역 메모리에 큐 형식의 데이터 구조를 만들어 들어오는 요청 순으로 저장하고 처리를 기다린다. 또한 기존 방식의 문제였던 사용자 영역 멀티 쓰레드 처리 구조를 커널 영역 단일 쓰레드 구조로 변화시켜서 스위칭 타임 및 메모리 오버헤드를 최대한 줄일 수 있는 구조를 가지고 있다. 이중 모델링에 공통적으로 들어가는 부분인 커널 구조의 개선점은 다음과 같다.

이 부분은 SSL세션이 필요한 요청이 서버에 들어왔을 때, SSL가속기의 처리를 거치고 난 후의 평문 요청을 처리하는 부분이다. 아파치의 사용자 영역 쓰레드 사용시 발생하는 문제를 없애기 위하여, 이 구조는 사용자 영역의 멀티 쓰레드대신 커널 영역의 단일 쓰레드를 사용하고 클라이언트 요청을 받아 단일 큐에 저장한다. CPU 당 하나의 커널 쓰레드가 큐에 저장된 차례로 하나씩 요청을 할당받고 기존에 캐시된 요청인지 판단하여 사용자 영역에 있는 아파치와의 통신을 하게 된다[2].

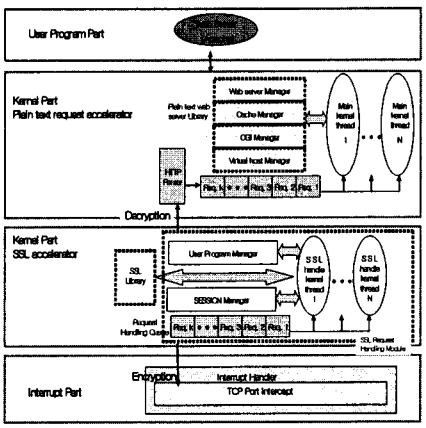

그림2 평문 요청 가속기 + SSL처리 가속기

3. 웹 서버 성능 관점에서 본 아파치와 커널 스레드 웹 가속기

3.1 Data Copies and Reads

매우 많은 응답을 필요로 하는 경우 웹 트랜잭션에서 memory 사이의 copy와 read를 줄이는 것은 성능향상에 지대한 영향을 미친다. 응답되어야 될 데이터가 파일시스템 캐시에 존재하는 사용자 모드에서의 웹서버는 데이터 copy를 피하기 어렵다. 데이터가 이미 사용자 모드의 주소영역에 매핑된 경우, BSD스타일 소캣은 NIC으로 데이터가 전달되기 전까지 한번 이나 그 이상의 데이터 copy가 필수적

이다. 데이터 copy가 제거된다 하더라도 checksum을 계산하기 위하여 데이터를 읽어야 하는 추가적인 오버헤드가 존재한다. 제안된 구조의 경우 커널 모드의 주소영역에서만 데이터를 다루고 바로 NIC의 버퍼와 데이터를 주고 받으므로 사용자 모드의 주소영역에서 복사하고 읽는데 발생하는 오버헤드를 줄일 수 있다.

3.2 Event Notification

Event Notification이란 클라이언트의 요청을 서버의 응답 태스크에 대하여 큐잉하는 것이다. 아파치 같은 경우 다중 스레드로 요청이 들어오는 즉시 스레드와 일대일 대응을 시키는 방법을 가지고 있고, Scala-AX의 경우 cpu 개수만큼의 스레드가 존재하여 커널단에 큐와 같은 데이터 구조를 만들어 그곳에 아직 처리되지 않고 있는 요청들을 큐잉한다. 다중 스레드의 경우 요청이 들어오는 매번 스레드를 만들어 할당하기가 부적절하므로 처음부터 스레드 pool을 운용하여 이미 만들어진 스레드에 요청을 할당하는 방식으로 스레드 생성시의 오버헤드를 극복한다. 그러나 이 방식 역시 해당 요청과 서버의 태스크 사이에 스케쥴링 오버헤드가 존재한다. scala-AX의 경우는 스레드 개수가 상대적으로 작으므로 스레드 들 사이의 컨텍스트 스위칭이나 요청과 스레드 사이의 스케쥴링 오버헤드를 줄일수 있다.

3.3 communication code path

대부분의 사용자 모드 웹서버의 경우, 새로운 커넥션을 맺거나, 요청의 데이터를 읽거나 상대방 주소의 정보를 얻는 경우, 파일 시스템에서 데이터를 읽거나 헤더나 데이터를 소켓에 쓸때를 위하여 시스템 콜 및 불필요한 소켓 계층의 코드를 사용한다.

4. 웹 가속 모듈의 CPN모델

각각의 토큰은 request에 의해 수행될 각각의 thread가 된다. 만약 동일한 place에 두 개 이상의 토큰이 존재한다는 것은 동일한 작업을, 즉 동일한 리소스 할당을 필요로 하는 작업을 두 개 이상의 thread가 수행한다는 것이 된다. 웹 가속기의 경우 request 하나당 thread 하나씩 사용하게 되므로 동시에 두개의 thread 를 사용할 수 없으므로, PN 전체를 통틀어 최초에 queue에 들어왔을때의 여러 개의 토큰이 있는 경우를 제외하고 어느 place에도 두 개 이상의 토큰이 존재할 수 없다. 즉 하나의 토큰이 작업을 끝낸후 또다른 토큰이 작업을 시작할 수 있다.

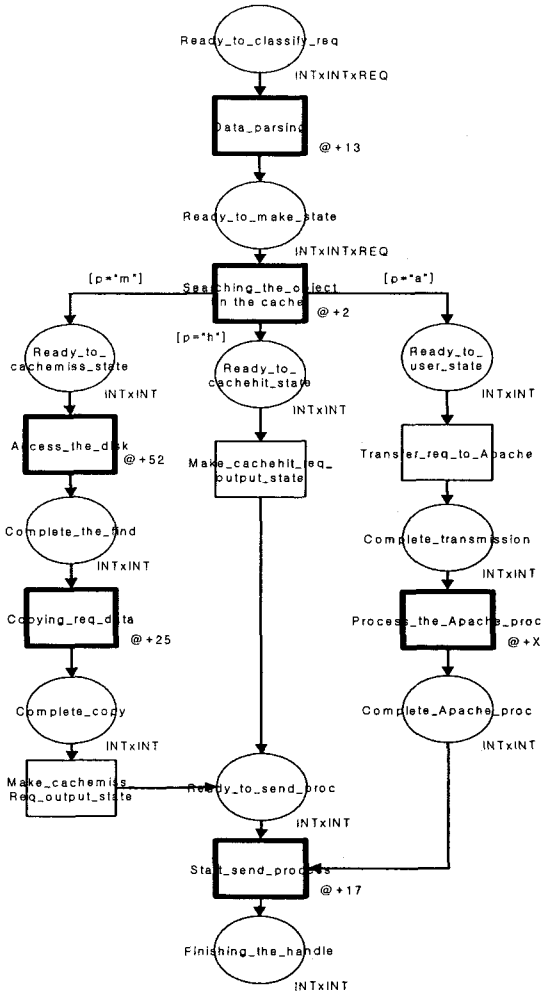

그림3 웹 가속모듈의 CPN 모델

토큰은 최초에 다음과 같은 multiple set의 color를 갖게 된다. (INT, INT, char) 여기서 첫번째 INT는 어떤 한정된 시간동안에 들어온 request들을 들어온 순서대로 번호를 할당하는 priority가 된다. 두번째 INT는 각각의 request가 요구하는 파일의 사이즈를 의미한다. 세번째 char 는 Petri-net 모델중에 논리적 분기를 요하는 부분이 있는데 그 부분에서는 각각의 request가 요구하는 파일의 속성이 다음 세 개중 한가지임을 이용한다. 웹 가속기의 cache hit일 경우 char는 "H", cache miss일 경우 "M", 그 나머지로 scala-AX가 다루지 못하는 동적 페이지의 경우 "A"로 할당을 한다.

Request는 Petri-net내부에서 각각의 요청 파일 속성에 따라 처리되는 routine이 다르다. File 사이즈에 따라 cache miss 의 경우 "Access_the_disk"

transition 의 time delay를 가변적으로 줄 수 있다. 이 경우 실제 scala-AX의 disk_access 함수 수행의 file 사이즈 별로 실제 수행 시간을 측정하여 그 값을 대입한다. Apache process의 경우, 해당 routine 중 "Process_the_Apache_proc"transition에 time delay 값을 cache miss 의 경우와 같이 직접 실행시켜 나오는 시간값을 vector로 주는 방법을 사용한다.

4. 시뮬레이션 요소들의 결정

임의의 퍼센티지로 Apache process를 요구하는 request, cache hit 되는 request, cache miss 되는 request 를 섞고 각각의 request 에 대하여 실제 연구된 바 있는 웹서버 request file 사이즈를 random 발생시켜 각각 400개부터 200개 단위로 1800개 까지 만든다. 그 후 PN의 최초의 place인 "Ready_to_classify_req"에 입력 벡터로 넣고, 시뮬레이션 하여 모두 처리된 후의 시간을 알아본다. 초당 처리된 요청수, 즉 작업 처리량 (throughput) = 입력 요청의 개수 / 처리하는데 걸린 시간으로 계산될수 있다. 위 CPN모델에서 timed 트랜지션으로 다음과 같은 것들이 있으며, 이것들은 실제 필요한 기능들을 코딩하여 해당 요청들을 수행하는 동안 각각의 트랜지션당 걸리는 시간을 리눅스 커널 함수인 do_gettimeofday()로 측정하였고, 반복횟수를 10000번씩 하는동안 90퍼센트 이상의 동일한 값을 나타내었다. 단, access_the_disk 와 copying request data 트랜지션의 경우는 요청 파일의 크기에 따라 다르기 때문에 평균값을 넣었다.

data parsing	13 us
searching the object	2 us
access the disk	52 us
copying request data	25 us
start send process	17 us

실제 인터넷 웹 트래픽은 대다수 정적 페이지 요청임을 가정할 때, 유저 모드의 아파치로 보내어 지는 요청이 없다고 가정할 수 있다. 또한 유저모드의 아파치가 수행되지 않는다면, 1개의 CPU를 갖는 서버의 경우, 한개의 커널 스레드만 작업에 할당할 수 있으므로 항상 모든 플레이스에는 한개의 토큰만이 존재할 수 있다. 따라서 총 수행시간의 계산은 각각 캐시 미스의 경우와 캐시 히트의 경우의 timed 트랜지션의 값들을 모두 더한 값으로 볼 수 있다. 아파치 hit ratio가 15%에서 30%사이이고 특히 클라이언트가 80개 이하일 경우 거의 15%에 있다는 벤치마킹을 사용하여[3], cache hit의 경우와 cache miss 의 경우를 나누고, 요청의 수를 늘렸다. cache hit의 경우, 13+2+17=32 us가 하나의 요청 수행에 필요한 시간이고, cache miss 의 경우, 13+2+52+25+17=109us가 하나의 요청수행에 필요한 시간이다. 이것의 비율을 15:85로 하여 계산하면, 0.01026개/us가 나온다. 이것은 이전 실제 가속 모듈 사용시 WebBench로 벤치마킹한 결과중 클라이언트 18개 이상일 경우 포화상태에 이를때의 throughput 이 0.0125개/us와 비교했을때 상당히 비슷한 결과임을 알 수 있다. 이 논문의 시뮬레이션의 경우, 각각의 timed 트랜지션마다 standardoutput 에 경과시간을 적어줘야 하고, 그것을 10000번씩 반복하게 되면서 커널의 작업 능력을 소모시킨 점을 가만하면, 약간의 throughput 감소를 예견할 수 있다.

5. 결론

이 논문에서는 커널 스레드를 이용한 웹 가속 구조를 생각하고, 그 소프트웨어 구조의 논리적인 부분들을 event 와 state로 나누었다. event의 경우, 트랜지션이 되고, state의 경우가 플레이스가 된다. 이러한 페트리넷의 기본에서 우리가 요구하는 동작특성을 좀더 표현하기 위하여 각각의 요청에 다른 color 값을 넣고 트랜지션에서 나가는 arc의 경우 논리적은 값을 첨가하였다. 이러한 CPN의 특성을 충분히 활용하여, 우리는 이 시스템이 단순히 논리적으로 문제가 생기는지의 여부, 즉 reachability를 떠나, 각각의 event에 따른 시간적 특성까지 살펴볼수 있었다. 그 결과 실제로 상용화 된 프로그램의 성능지표를 그대로 얻음으로서 기존에 제안한 웹 가속 소프트웨어의 논리적인 흐름을 이해할 수 있었고, 앞으로 소프트웨어의 분석 및 설계를 하는데 응용할 수 있는 방법론을 발견할 수 있게 되었다.

참고문헌

[1] Matt Welsh, et al. "SEDA: An Architecture for Well-Conditioned, Scalable Internet Services" Proc. ACM Symposium on operating systems principles, vol. 35, 2001

[2] J. Park, H. Lim and H. Kim, Development of kernel thread web accelerator, IEE Electronic Letters, vol.38, no.13, June 2002-09-09, pp.672-673

[3]http://www.cs.wisc.edu/~cao/papers/cao-wpb/node10.html