커널 스레드 웹 가속기의 분석

황 준*, 남의석**, 민병조*, 김학배*
*연세대학교 전기전자공학과 **극동대학교 정보통신학부
전화 82-2-2123-2778 / 팩스 82-2-2123-3780

Analysis of Kernel-Thread Web Accelerator

June Hwang*, EuiSeok Nahm**, Byungjo Min*, Hagbae Kim*
*Dep. of Electrical and Electronic Eng. Yonsei University
**School of Information & Telecommunication, Far East University
hbkim@yonsei.ac.kr

Abstract

The surge of Internet traffic makes the bottleneck nowadays. This problem can be reduced by substituting the media of network, routers and switches with more high-performance goods. However, we focused radically the server performance of processing the service requests. We propose the method improving performance of server in the Linux kernel stack. This accelerator accepts the requests from many clients, and processes them using not user threads but kernel thread. To do so, we can reduce the overhead caused by frequent calling of system calls and the overhead of context switching between threads. Furthermore, we implement CPN(Coloured Petri Net) model. By using the CPN model criteria, we can analyze the characteristics of operation times in addition to the reachability of system. Benchmark of the system proves the model is valid.

1. 서론

최근 인터넷 인구의 폭발적 증가로 인하여 네트워크에 웹 트래픽이 폭주하고 있다. 기존에는 사용자 측면에서 느린 웹 서비스 응답을 받는 주된 이유가 클라이언트 측의 낮은 대역폭이 원인이었다. 그러나 현재는 NIC(Network Interface Card)나 전송매체가 저가격화 되어 가고 있고 그에 따라 클라이언트 측면에서 좀 더 고성능의 NIC를 사용하고 있으며, 네트워크 인프라의 물리 매체 역시 저가격에 힘입어 좀 더 고성능, 고사양의 매체를 사용하게 되었다. 이에 따라 네트워크 상에서 트래픽의 병목현상을 유발하게 되는 주된 이유는 기존처럼 네트워크의 대역폭이 아니라 서비스하는 서버의 처리 성능 저하로 볼 수 있다.

이러한 이유들에 의하여, 이미 서버의 성능향상에 대한 많은 연구들이 있어왔다. 우선, 서버로 들어오는 네트워크의 부하 분산을 위한 별도의 독립형 서버 장비의 추가를 대안으로 제시한 경우가 있다. 또한, 서버 앞단에 별도의 캐싱 서버를 두는 방법이 있는데, 일종의 프록시 서버의 개념이 된다. 또, 웹 서버의 요청에 따른 서버 자체의 소비 자원을 각각 차별화 하여, 해당 자원의 사용 상태에 따라 요청을 큐에서 수락제어 하는 방법이 있다. 그러나 위와 같은 방법들은 별도의

장비의 구입, 관리 그리고 운영이라는 추가적인 비용 및 작업을 원하는 방법이었다. 또한 이러한 방법들은 사용자 수준의 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 갖는 SCALA-AX를 제안한다.

2. 기존 웹 서버의 동작 구조 및 문제점

아파치는 오늘날 가장 많이 쓰이는 웹 서버 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 읽어 와서 캐시하고 그것을 클라이언트로 보내게 된다. 각각의 요청에 대해 worker 쓰레드마다 별도의 독립성을 가지고 요청을 수행한다. 아파치는 사용자 레벨의 응용프로그램이므로 사용자 쓰레드를 사용하게 된다. 사용자 쓰레드를 이용할 때, 시스템콜을 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. [20]에서는 실제로 user-level 의 스레드를 사용하여 클라이언트의 매 요청을 각각 하나의 스레드에 할당하였다. 그리고 그 요청의 빈도를 높여가며 작업처리량(throughput)을 측정한 결과, 특정 요청의 개수가 동시에 스레드에 할당 되었을 때 까지는 처리량이 증가하였으나, 그 이후로는 급격한 성능감소를 보였다. 상대적으로 편리한 프로그래밍 방법임에도 불구하고, threading 과 관련된 오버헤드들 - cache와 TLB (Translation Lookaside Buffer) 미스, 스레드 사이의 스케줄링 오버헤드, 공유 리소스의 lock 경쟁 등- 이 스레드의 개수가 커질 때 마다 성능의 심각한 감쇠를 가져오게 된다[21].

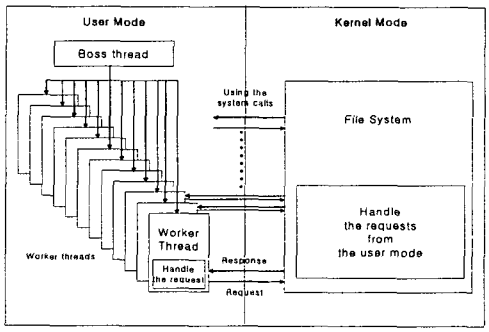

그림29 아파치의 내부 동작구조

3. 개선된 웹 가속기의 구조

이 부분은 필요한 요청이 서버에 들어왔을 때, 처리하는 부분이다. 아파치의 사용자 영역쓰레드 사용시 발생하는 문제를 없애기 위하여, 이 구조는 사용자 영역의 멀티 쓰레드 대신 커널영역의 단일 쓰레드를 사용하고 클라이언트 요청을받아 단일 큐에 저장한다. CPU 당 하나의 커널쓰레드가 큐에 저장된 차례로 하나씩 요청을 할당받고 기존에캐시된 요청인지 판단하여 사용자 영역에 있는 아파치와의 통신을 하게 된다. 또한 개발된 가속기는 기존에 있던 아파치 내의 캐시에 비하여 월등한 성능의 캐시 알고리즘을 별도로 채택하였다. 여기서 월등한 성능이란, 아파치의 캐시는 사용자 레벨의 메모리 주소 영역에 존재하지만, SCALA-AX의 캐시는 커널 영역의 메모리 주소에 있기 때문에, 기존처럼 사용자 영역 메모리에서 캐시의 객체를 읽어와 커널 영역 메모리로 복사하는 과정을 거치지 않아도 되기 때문에 생기는 것이다. 그로 인해 본 가속기는 캐시된

객체들의 placement/ replacement등 웹 서버의 객체 관리에 관한 성능을 향상시킨다. 또한 HTTP1.1의 지속연결과 연속 요청을 완전 지원한다.

프로그램 구조중 라이브러리 함수는 다음과 같이 구성되어 있다. 가속기가 별도로 처리하지 못하는 객체 즉 PHP, JSP등의 동적 객체를 아파치로 넘겨 처리를 요구하는 웹 서버 매니저, 요청된 객체가 메모리에 저장되어 있는가를 판단하고, 그 객체의 원본과의 동기화를 담당하는 캐시 매니저, 동적 객체들 중에서 본 가속기가 캐시 할 수 있는 종류의 객체인 CGI(Common Gateway Interface)를 별도로 담당하는 CGI매니저, 가상 웹 서비스를 담당하는 가상 호스트 매니저가 있고 TCP 단에서의 데이터를 가로채어 HTTP형식으로 해석하는 HTTP 해석기가 있다.
그림 2는 가속기와 그 내부의 구성요소간의 동작에 대하여 개략적으로 나타내었다.

이 구조는 N개의 프로세서를 갖는 SMP (Symmetric MultiProcessing) 머신의 경우를 나

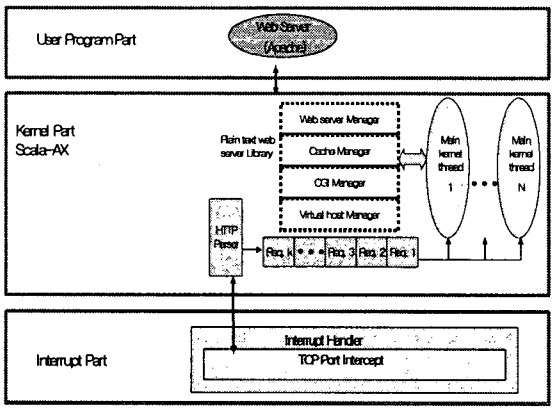

그림 2 SCALA-AX 동작 구조

타낸다. 우선 특정 파일을 요구하는 요청이 이 서버에 도착했을 때, 소프트웨어 인터럽트에 의하여 TCP 부분 인터셉터에서 이를 가로채고 이것을 HTTP 분석기가 분석하여 큐에 저장하고 그 차례로 요청당 하나씩 N개의 커널 쓰레드가 할당되어 웹 서비스를 수행하게 된다. 그 쓰레드들은 커널에 존재하는 함수 및 라이브러리를 이용하여 CGI 작업 처리를 하던가, 캐시 매니저를 이용하여 캐시에 요청 대상이 있는지 확인하고 없으면 직접 디스크를 액세스 하여 읽어 오고, 있으면 캐시에서 가져와 클라이언트로 보내게 된다.
다음은 기존 웹 서버의 요청 처리 방식과 개선된 방식을 비교한 것으로서 사용자 레벨의 멀티스레드와 커널 레벨의 싱글 스레드의 차이를 보여준다.

그림3 에서처럼, 기존에는 들어오는 요청의 개수만큼의 처리 프로세스 생성으로 인해 시간 및 메모리의 오버헤드가 많이 발생하고, 동일한 우선순위에 의하여, 타임 슬라이스를 기준으로 지속적인 스케쥴링을 하여, 스레드 간의 스위칭으로 인한 오버헤드가 존재하였으나, 개선된 방식에서는 요청을 처리 큐에 두고 CPU당 하나의 커널 쓰레드가 독점적으로 웹 요청 처리를 함으로써 스위칭시의 오버헤드가 거의 존재하지 않음을 알 수 있다.

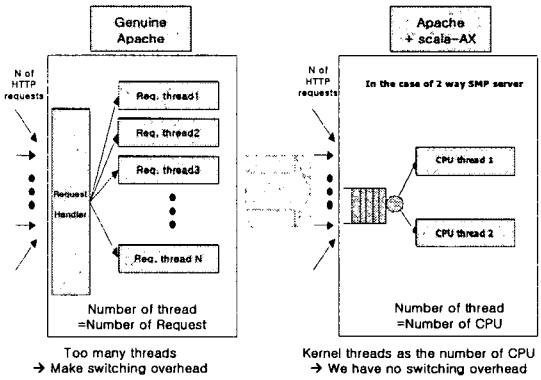

그림 3 웹 서버의 처리방식 비교

4. Coloured Petri-Net을 이용한 SCALA-AX 모델링

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. SCALA-AX의 경우 request 하나당 thread 하나씩 사용하게 되므로 동시에 두개의 thread 를 사용할 수 없

으므로, PN 전체를 통틀어 최초에 queue에 들어왔을때의 여러 개의 토큰이 있는 경우를 제외하고 어느 place에도 두개 이상의 토큰이 존재할 수 없다. 즉 하나의 토큰이 작업을 끝낸후 또다른 토큰이 작업을 시작할 수 있다.

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. SCALA-AX의 cache hit 일경우 char는 "H", cache miss 일경우 "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 routine중 "Process_the_Apache_proc"transition에 time delay 값을 cache miss 의 경우와 같이 직접 실행시켜 나오는 시간값을 vector로 주는 방법을 사용한다.

scala-AX의 동작을 구분하여 모델을 만들면 그림 13과 같다. PN 모델의 트랜지션 중 특히 timed 트랜지션을 직접 구현하고, test suite 파일을 이용하여 각 트랜지션의 파일 속성에 대한 파라미터를 구한다. 각각의 timed 트랜지션에 대한 값들은 테스트 파일의 벤치 마킹을 통하여 구해질 수 있다.

Data_parsing : 13(us)
Searching_the_object : 2(us)
Make_cache : y=0.007x + 70.01 (us)
여기서 x는 파일사이즈(byte),
Start_send_proc : 50.8 (us)

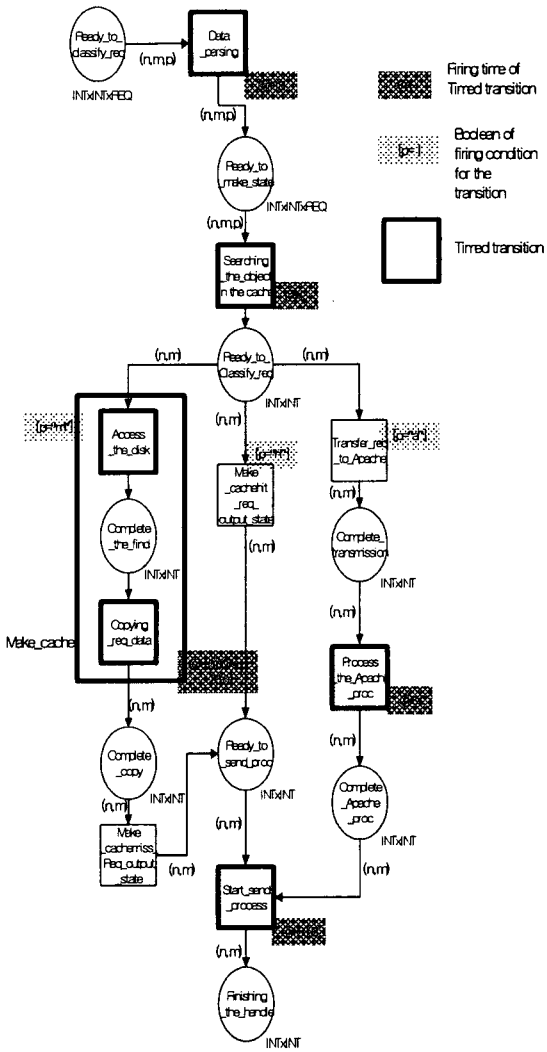

그림 13 SCALA-AX의 완성된 CPN 모델

5. CPN 모델 유효성 검증

SCALA-AX를 구동 시킨 상태에서 각각의 test suite를 요청하여 보고, 부분의 시간이 아닌 전체 수행 시간을 측정한다. 전체 수행 시간의 측정으로 CPN 모델의 유효성을 검증 할 수 있으며, 다음에 나오는 여러종류의 테스트 파일을 복합적으로 벤치 마킹한 필드 테스트와의 유효성을 검증

할 수 있다. 본 연구에서 end-to-end user 지연 시간의 벤치마킹시 SCALA-AX의 성능을 평가하기 위해서, 800MHz 의 dual PC processor를 사용하였다. 웹서버에 대한 요청은 zdnet의 web bench 3.0을 사용하였으며, 각각 gigabit LAN 환경에서 총 24대의 테스트 클라이언트 서버를 통해 이루어졌다. 웹서버의 성능향상을 보여주는 기준으로 초당처리 요청수와 throughput을 측정하였다. 그림 16은 테스트 클라이언트당 초당 처리 요청수를 보여주고 있다. 아파치만으로 웹 서비스를 하였을 경우, 웹서버는 약 초당 1000개의 요청을 수행하였으며, SCALA-AX를 탑재하여, 웹 가속 및 캐싱 서비스를 하였을 경우, 대략 초당 5000개의 요청을 수행하였다. 아파치만으로 이루어진 웹서버에 비하여 SCALA-AX가 탑재된 웹서버는 초당처리 요청수 부분에서 약 5배 이상의 성능향상을 보임을 알 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 $T_{parcing}$ + $T_{searching}$ + $T_{startsending}$ = 13 + 2 + 50 = 65 (us)가 하나의 요청 수행에 필요한 시간이고, cache miss 의 경우, $T_{parcing}$ + $T_{searching}$ + $T_{makecache}$ + $T_{startsending}$ = 13 + 2 + 77 + 50 = 142 (us) (1)가 하나의 요청수행에 필요한 시간이다. 여기서 $T_{makecache}$는 0.007 X filesize(byte) + 70 = 0.007 X 1000 + 70 = 77 (us)(2)으로 계산된다. 즉, WebBench 에서 사용된 test file의 평균값이 약 1000 byte 였다.

또한 ping 으로 해당 네트워크, 해당 서버와 클라이언트들의 사이에서 걸리는 시간을 측정한 결과, 한 개의 IP 데이터그램의 라운드 트립 타임(RTT)이 byte당 0.2us의 시간이 걸렸다. 따라서 평균 1000byte의 파일이 네트워크를 돌아 다닌다고 보면, 하나의 파일 요청에 대하여 순수 SCALA-AX의 처리 이외에 200us의 시간이 더 붙는 것이 된다. 100개의 파일을 기준으로 cache hit : cache miss = 15 :85이므로, 그 100개의 요청이 네트워크를 제외하고 순수하게 처리되는 데만 걸린 총 시간 T는 (1),(2)에서

$$65\,(us) \times 15 + 142\,(us) \times 85 = 13045\,(us) \quad (3)$$

가 된다. 따라서 $T_d = \frac{T}{2} = 6522.5\,(us)$가 된다.

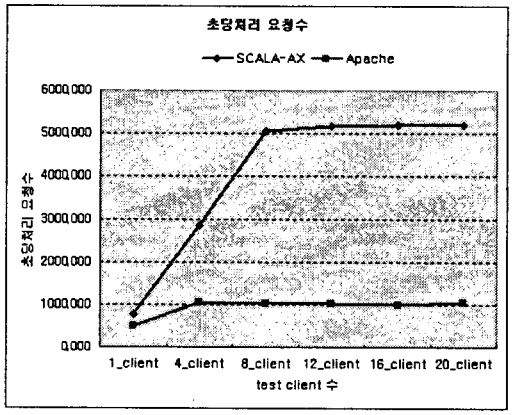

그림 16 테스트 클라이언트 당 초당처리 요청

또한, 각각의 파일이 네트워크를 통하여 지연되는 시간, 좀 더 정확히 말하자면, 두개의 스레드로 처리된 파일 역시 concurrent 하게 두개씩 네트워크를 타고 나가게 됨으로 100개와 50개 사이의 어떤 값을 가질 것이다. 대략 65%정도로 네트워크 소요 시간을 잡을 수 있다. 이 점을 감안하면,

$$6522.5\,(us) + 200\,(us) \times 65 = 19522.5\,(us)$$

가 된다. 100개의 요청 처리시 이시간이 걸리므로, 1개의 요청처리시에는 195.225us가 걸리며, 5122.23개의 초당처리 요청수를 보인다.

이것은 그림 7에서 보는 바와 같이 8개 이상의 클라이언트가 요청을 보낼 경우, SCALA-AX의 처리량이 포화된다. 이때 초당 처리 요청수가 약 5100을 가리키고

있다. 이것은 CPN 모델을 통한 시뮬레이션 결과와 실제 WebBench를 통한 벤치 마킹 결과가 거

의 유사함을 알 수 있다.

6. 결론

이 논문에서는 커널 스레드를 이용한 웹 가속 구조를 생각하고, 그 소프트웨어 구조를 CPN으로 나타내고, 모델링으로부터 유추되는 값들을 필드 데이터와 비교하여 모델의 유효함을 입증하였다. 커널 스레드 웹 가속기는 사용자 수준의 프로그램이 아니라 커널에 존재하며 모듈로서 기능을 하는 프로그램이다. 기본이 되는 특징으로 기존 아파치의 경우 사용자 레벨의 멀티스레드를 사용하는 반면에, scaoa-AX는 커널 레벨 CPU갯수와 일치하는 스레드를 사용한다는 점이다. 게다가 캐시기능을 사용하여, 사용자 영역의 메모리를 액세스 하지 않고도 요청시 원하는 파일을 서비스 해줄수 있다.우리는 웹 가속기의 동작을 단순화 하여 각각의 event를 트랜지션으로 생각하고, event 전후의 상태를 나타내는 부분들은 플레이스로 생각하여 웹 가속기의 CPN 모델을 만들었다. 큰 줄기로 캐시된 요청 파일, 캐시되지 않은 요청 파일들에 대하여 동작하는 부분들이 다르고 파일 크기와 firing time과도 상관관계가 있음을 알아냈다. 이 상관관계를 이용하여, 모델의 트랜지션의 파라미터들을 알고, 이 모델을 이용하여 각각의 테스트 파일들을 실제로 요청해보았다.

참조문헌

[20] M. Welsh, D. Culler, and E. Brewer, "SED A :An Architecture for Well-Conditioned, Scalable Internet Services," Proc. ACM Symposium on operating systems principles, vol.35, 2001

[21]D.P. Bovet, M. Cesati, "Understanding the LINUX KERNEL," 2nd edition, O'Reilly & Associates, 2003

[3] J. Almeida, P. Cao, "Measuring Proxy Performance with the Wisconsin Proxy Benchmark," http://www.cs.wisc.edu/~cao/papers/cao-wpb/