아파치 웹 서버에서의 다중 쓰레드 풀 활용 기법 분석

(Analysis of Multi-thread Pool Utilization Scheme on the Apache Web Server)

전 흥 석 † 이 승 원 †† 강 현 규 †††
(Heung Seok Jeon) (Seung Won Lee) (Hyun Kyu Kang)

요 약 웹 서버 혹은 웹 애플리케이션 서버는 급증하는 웹 사용자들의 요구에 효율적으로 대처하기 위하여 일반적으로 다중 쓰레드 모델을 적용하고 있다. 그러나 이러한 다중 쓰레드 모델이 새로운 웹 환경의 특정한 상황에서 다중 프로세스 모델에 비해 오히려 더 나쁜 성능을 보이는 경우가 있다. 본 논문에서는 이러한 문제를 해결하기 위하여 두 가지 접근 방법을 통해 다중 쓰레드 모델의 성능 저하에 대한 원인을 분석한다. 그 중 첫 번째로, 다양한 응용 환경에서의 다중 쓰레드 모델과 프로세스 모델을 비교한다. 두 번째로는 효율성을 위하여 일반적으로 제공되는 프로세스/쓰레드 풀 모델에서 동적인 지시자들의 설정 값에 의한 영향을 분석한다. 본 논문에서는 자체 제작한 웹 클라이언트 시뮬레이터와 아파치 웹 서버 2.0을 연동하여 이러한 실험들을 진행하고 결과 및 분석 내용을 제시한다.

키워드 : 웹 서버, 다중 쓰레드, 다중 프로세스, 지시자

Abstract Web servers or web application servers, in general, adopt multi-thread model for efficient handling of many user requests. However, the multi-thread model always does not show the better performance than multi-process model. Sometimes, in a certain specific case, it can show worse performance than multi-process model. In this paper, to trace the cause of the decreased performance of multi-thread model, we experiment and analyze the performance of the multi-thread model by using two approaches. At first, we compare the performance of the multi-process model and multi-thread model for various application environments. Second, we observe the effects of variations of web server's dynamic directives, which are used to increase the flexibility of the web server for various system environments. For the experiments, we integrated a web client simulator, which was written by us, with the Apache 2.0 web server. This paper shows and analyze the results of the experiments.

Key words : web server, multi-thread, multi-process, directives

1. 서 론

초기의 웹 서버나 웹 애플리케이션 서버들은 다중-프로세스 모델을 사용하여 다중 클라이언트로부터의 동시 요청들을 처리하여 왔다. 그러나 시스템의 전체 사용자 수가 증가하면서 다중 프로세스 모델은 새로운 프로세스의 생성이나 프로세스간의 문맥 전환 등을 위한 오버헤드가 큰 부담이 되고 있다.

† 비 회 원 : 건국대학교 컴퓨터응용과학부 교수
hsjeon@kku.ac.kr
†† 비 회 원 : 건국대학교 컴퓨터정보통신공학부
swlee@db.konkuk.ac.kr
††† 종신회원 : 건국대학교 컴퓨터응용과학부 교수
hkkang@kku.ac.kr
논문접수 : 2003년 10월 28일
심사완료 : 2004년 10월 7일

이를 해결하기 위하여 최신의 웹 서버들은 다양한 개선책들을 시도하고 있다. 그 중에 하나가 다중 쓰레드 모델을 활용하는 것이다. 쓰레드는 한 프로세스가 보유하고 있는 주소 공간(address apace)을 쓰레드들 간에 공유함에 의해서 쓰레드의 생성 및 문맥 전환의 부담을 최소화하는 장점을 가지고 있다. 이러한 장점을 활용하여 많은 웹 서버 혹은 웹 애플리케이션 서버들이 다중 쓰레드 모델을 적용하고 있는 추세이다. 예를 들어, 대표적인 공개 웹 서버인 아파치 웹 서버도 1.3 버전에서는 다중 프로세스 모델을 사용하였지만, 2.0 버전에서는 다중 쓰레드 모델을 적용하고 있다[1,2].

그러나 다중 쓰레드를 사용하는 것이 항상 좋은 결과만을 가져오는 것은 아니다. 최근의 한 연구 결과에 따르면 다중 쓰레드를 사용하는 것이 오히려 다중-프로세

스를 사용하는 시스템에 비해 성능이 나빠질 수 있다는 연구 결과를 보여주고 있다[3]. 그 내용을 잠시 인용하자면, 그림 1은 해당 논문에서 제시한 아파치 1.3과 2.0의 다양한 문서에 대한 요청을 처리하는데 소요된 응답 시간을 보여준다. 그림 1에서 보면 멀티 쓰레드를 기반으로 하는 아파치 2.0의 응답 시간이 동적인 문서에 대해 아파치 1.3보다 더 나쁜 결과를 가져오는 경우가 발생하였다. 이것은 쓰레드를 사용한다고 해서 항상 더 좋은 성능을 가져오는 것은 아님을 확인해 준다.

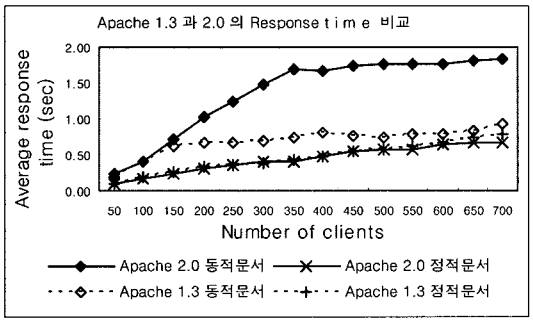

그림 1 Apache1.3과 2.0의 Response time 비교

이러한 결과에 대한 원인이 해당 논문에서는 프로세스 모델과 쓰레드 모델간의 문맥전환의 수의 차이에 있다고 분석하였다. 표 1은 다중 프로세스 모델을 적용하고 있는 아파치 웹 서버 1.3과 다중 쓰레드를 사용하고 있는 아파치 웹서버 2.0에서 다양한 형태의 정적인 문서와 동적인 문서의 혼합된 형태에 대하여 두 버전의 시스템에서 발생하는 프로세스 혹은 쓰레드들의 문맥전환 수를 측정한 결과이다.

일반적으로 다중 프로세스와 쓰레드의 각각의 문맥 전환에 요구되는 오버헤드는 서로 다른 것으로 알려져 있기 때문에 이를 보정하기 위하여 그림1에서는 '동등비율'이라는 항목을 추가로 보여준다. 동등 비율이라는 것은 그림 1의 아파치 1.3과 2.0의 문맥 전환에 요구되는 전체 오버헤드가 같다고 가정했을 경우에 쓰레드의 프로세스에 대비한 문맥전환 오버헤드 비율을 의미한다. 예를 들어 설명하자면, 단계 1의 경우에 아파치 1.3의 문맥전환 수는 50,902이고 아파치 2.0의 문맥전환 수는 81,282일 경우에 쓰레드가 프로세스에 비해 문맥전환을 위한 오버헤드가 0.02(2%)에 불과할 경우 두 시스템의 문맥 전환을 위한 전체 오버헤드가 같게 된다는 것이다.

실험 결과를 보면 아파치 1.3에 비해 아파치 2.0의 경우에 전체 문맥 전환의 수가 대체로 매우 많음을 알 수 있다. 물론 쓰레드의 문맥 전환에 요구되는 오버헤드는 프로세스의 문맥 전환을 위한 오버헤드에 비해서는 작다고 알려져 있다. 문제는 이 작은 정도가 생각보다 그리 크지 않다는 것이다. 관련 연구 [1]에 따르면 쓰레드의 문맥 전환을 위한 오버헤드는 프로세스보다 약 70% 정도의 감소 효과가 있다고 알려져 있다. 이 기준에 따르면 그림 1의 동등 비율이 0.7 보다 작은 경우에는 쓰레드를 사용하는 것이 오히려 성능이 좋지 않을 수 있다는 것이다. 이러한 경향은 정적 문서와 동적 문서의 비율이 7:3인 단계 8에서부터 시작하여 동적 문서의 비율이 많아질수록 더 심해진다.

그러나, 이러한 문맥 전환이 왜 멀티 쓰레드 모델에서 상대적으로 많이 발생하게 되는지에 대한 원인은 정확히 알려져 있지 않다. 최근의 많은 웹 애플리케이션들은 정적인 문서보다 CGI를 통한 동적인 문서를 많이 취급하고 있으며, 그 경향은 점점 더 커져갈 것이다. 그러므로 다중 쓰레드의 효율적인 활용을 위해서 이 문제의 원인을 밝히는 것은 매우 중요한 문제인 것이다.

따라서 본 논문에서는 이에 대한 원인을 밝히기 위해 다음과 같이 두 가지 형태로 분류하여 실험을 진행한다. 첫 번째는 다중 쓰레드와 프로세스 모델이 웹 서비스에 이용되었을 때 적용되는 응용 프로그램의 특성 및 환경

표 1 Apache 1.3과 Apache 2.0의 문맥 전환의 횟수 비교를 통한 성능 분석 결과

단계	문서비율		Apache 1.3	Apache 2.0			동등비율
	정적문서(%)	동적문서(%)	프로세스	프로세스	쓰레드	합계	
1	0	100	50902	50214	31068	81282	0.02
2	10	90	48336	49995	27107	77102	--
3	20	80	47754	44762	31236	75998	0.10
4	30	70	49503	44203	32690	76893	0.16
5	40	60	49490	42461	33827	76288	0.21
6	50	50	48351	39351	33218	72569	0.27
7	60	40	42473	30681	32990	63671	0.36
8	70	30	40721	26980	30960	57940	0.44
9	80	20	45229	22906	30954	53860	0.72
10	90	10	47018	22862	30040	52902	0.80
11	100	0	43842	13965	40377	54342	0.74

이 어떠한 영향을 미치는지를 알아보기 위한 실험을 진행한다. 두 번째는 아파치 2.0 모델의 성능을 극대화하기 위해 제공된 다양한 동적인 지시자(directive)들이 어떠한 영향을 미칠 수 있는 가를 알아보기 위한 실험을 진행한다. 그 중 첫 번째 실험을 위해서 다시 두 가지의 서로 다른 환경을 설정하였다. 그 중 하나는 웹 서버에 접속하는 동시 접속자의 수가 급증하는 환경이고, 다른 하나는 인터넷을 비롯한 웹 기술이 바이오나 그리드 컴퓨팅 등의 환경과 같이 서버에서의 계산 시간이 급증하는 환경이다.

본 연구에서는 이에 대한 실험을 통해 첫 번째 상황은 멀티 쓰레드의 성능에 거의 영향을 미치지 않음을 확인하였고, 두 번째 실험인 아파치 서버의 다양한 지시자의 변화에 의해 다중 쓰레드 모델이 다중 프로세스 모델보다 나쁜 성능을 가져 올 수 있음을 확인하였다. 자세한 내용은 해당 절에서 설명한다.

본 논문의 나머지는 다음과 같이 구성되어진다. 2절에서는 본 논문에서의 주요 관심 대상인 아파치 2.0 웹 서버의 구성 및 동작 원리에 대해 분석한 결과를 기술한다. 3절에서는 다중 쓰레드 모델의 상대적 성능 저하에 대한 원인을 분석하기 위한 연구 진행 방향에 대해 정리하고 4절에서는 실험 환경과 실험 결과에 대해 제시하고 분석한다. 5절에서 간단히 결론을 맺고 향후 연구 과제를 제시한다.

2. 아파치 2.0 웹 서버의 구성 및 지시자 분석

이 절에서는 본 논문에서 실험에 활용할 아파치 2.0 웹 서버의 다중 프로세스 및 다중 쓰레드 관련 부분에 대한 동작에 대해 분석한 내용을 기술한다. 아파치 웹 서버 실행 시 선택할 수 있는 세 가지 실행 모드 및 성능 최적화를 위해 제공되는 다양한 지시자의 의미 및 설정 방법에 대해 자세히 분석한다.

2.1 아파치 2.0 웹 서버

아파치 웹 서버는 NCSA httpd 1.3을 기반으로 개발된 웹 서버이다. 전 세계 웹 서버 시장의 60% 이상을 점유하고 있는 아파치는 Apache HTTP Server Project에서 지속적으로 업그레이드 버전을 발표해오고 있다[1,2]. 현재 발표된 아파치의 최신 버전은 2.0.47 이며 본 논문에서 사용한 것은 2.0.45 버전이다.

아파치 2.0은 웹 시스템의 다양한 사용 환경에 따라 달라지는 실행 조건에 대응하기 위하여 다양한 선택적 실행 방법을 가지고 있다. 그 중 유닉스 플랫폼에서 사용하는 대표적인 것으로 worker, prefork, 그리고 perchild 모듈이 있다. 이러한 모듈들은 컴파일 시 결정해야 하며 한번에 하나의 모듈만 컴파일 될 수 있다. 컴파일한 후 실행에 관련된 프로세스와 쓰레드의 수 등 여러 가지 설정들은 환경 설정 파일(httpd.conf)의 지시자(directive)를 변경함으로써 가능하다[1,2].

Worker와 prefork 모듈들은 각각 확장성(많은 요청에 대해 유연하게 대처할 수 있는 능력)이 요구되는 사이트, 안정성과 오래된 소프트웨어와의 호환성이 필요한 사이트에 주로 사용되고, perchild 모듈은 다른 사용자 계정으로 여러 호스트를 서비스 하는 것에 사용 된다[1]. 이와 같은 모듈들 중 본 논문에서는 두 가지 이유로 worker 모듈을 사용한다. 그 첫 번째 이유는, 사용자들의 급격한 증가에 대처 할 수 있는 능력을 높이기 위한 목적으로 개발된 모듈이라는 점이고 두 번째로는, worker 모듈의 지시자를 변경함으로써 멀티 쓰레드와 멀티 프로세스 구조를 동시에 실험할 수 있다는 것이다. 다음 절에서 두 번째 이유에 관해서 자세히 설명한다.

2.2 Worker 모듈의 지시자 연구

Worker 모듈은 멀티 프로세스와 멀티 쓰레드의 혼합형 동작방식을 사용한다. Worker 모듈과 관련된 지시자들은 StartServers, ThreadsPerChild, MinSpareThreads, MaxSpareThreads, ServerLimit 그리고 MaxClients가 있다. 프로그램이 실행되면 주 프로세스는 StartServers에 설정되어 있는 수만큼의 자식 프로세스들을 생성한다. 자식 프로세스들은 ThreadsPerChild의 설정에 따라 각각 쓰레드들을 생성한다. 이 쓰레드들이 실제 요청에 대한 서비스를 수행하며 이 지시자를 통해서 하나의 프로세스가 하나의 요청을 처리하는 멀티 프로세스 구조, 혹은 하나의 프로세스가 다수의 쓰레드를 보유함으로써 여러 개의 요청을 처리하는 멀티 쓰레드 구조 등의 다양한 환경을 설정할 수 있다.

현재의 쓰레드들 중 요청에 대한 서비스 중에 있지 않은 쓰레드들이 스페어(spare) 쓰레드 이며, 프로그램을 실행한 후 어떠한 요청도 없다면, StartServers * ThreadsPerChild가 스페어 쓰레드 수가 된다. 만약 이 때 요청이 들어오면 스페어 쓰레드의 수가 줄게 될 것이고, MinSpareThreads보다 적은 수의 스페어 쓰레드가 남게 된다면, 주 프로세스는 자식 프로세스를 추가로 생성한다. 그리고 다시 그 자식 프로세스는 ThreadPerChild에 설정되어 있는 수의 쓰레드들을 생성한다. 이런 방식으로 생성될 수 있는 자식 프로세스의 총 수는 ServerLimit까지 가능하다. 요청에 대한 응답을 마쳐서 스페어 쓰레드가 MaxSpareThreads 보다 많게 된다면 자식 프로세스 중 하나를 제거한다. 이렇게 해서 MinSpareThreads와 MaxSpareThreads사이의 범위로 스페어 쓰레드들을 유지한다. 그리고 만약 어떤 자식 프로세스가 처리한 요청의 수가 MaxRequestsPerChild 만큼이 되면 그 자식 프로세스를 제거한다. 이 지시자가 0으로 설정되어있으면 처리한 요청의 수를 제한하지 않

는다. MaxClients는 동시에 서비스 가능한 클라이언트의 수를 설정하는데 쓰이는 지시자 인데, 이것은 ServerLimit * ThreadsPerChild까지 제한된다. 만약 연결 요청이 MaxClients 보다 많이 들어온다면 ListenBacklog (대기하는 queue의 크기를 정하는 지시자)의 수 까지 대기(queue)하게 된다.

3. 연구 진행 방향 및 실험 환경

3.1 실험 내용

다중 쓰레드 모델이 다중 프로세스 모델에 비해 성능이 저하되는 원인을 밝히기 위해 본 연구에서는 여러 가지 형태의 실험 및 분석을 진행한다. 본 연구를 통해 시도된 실험들을 정리하자면 크게 다음과 같은 두 가지 형태로 분류 되며, 좀더 자세하게 세 가지 형태로 분류할 수 있다.

첫째는 성능 저하의 원인이 외부의 환경의 변화에 의해 추가적인 오버헤드의 발생으로 인한 것인가에 대한 실험이다. 이 실험에서는 두 가지의 소주제로 분류하여 실험한다.

그 중 하나는 웹의 사용자가 급증하여 웹 서버에 대한 동시 접속자의 수가 급증하는 경우에 멀티 쓰레드 구조의 서버와 멀티 프로세스 구조의 서버에 미칠 수 있는 영향에 대한 실험이다. 이 실험은 동시 접속자의 수를 변화하면서 다중 쓰레드 모델과 다중 프로세스 모델의 성능의 변화를 관찰한다.

다른 하나는 차세대 컴퓨팅 환경이 성능에 미치는 영향을 분석한다. 구체적으로 말하자면, 차세대 환경이란 인터넷 및 웹에 대한 활용도가 증가하여 웹 환경이 그리드 컴퓨팅이나 바이오 애플리케이션 등 대규모와 고성능의 컴퓨팅 능력을 요구하는 환경을 의미한다. 이러한 환경에서는 애플리케이션의 특성 상 프로세서의 활용 율이 매우 높아지게 되며, 이러한 변화가 다중 쓰레드의 모델에 어떠한 영향을 미치는 지에 대해 시뮬레이션을 통해 분석한다.

마지막으로 위와 같은 외부적인 요인이 아닌 아파치 웹 서버의 자체적인 문제로 인해 발생할 수 있는 성능 저하 가능성에 대해 실험을 진행한다. 2절에서 기술한 것과 같이 아파치 웹서버 2.0에서는 다중 프로세스와 다중 쓰레드 모델을 혼용하여 사용할 수 있도록 되어있다. 사용자의 요청에 응대하는 프로세스 혹은 쓰레드의 수 및 시스템 부하에 따른 환경 설정을 다양한 지시자들을 통해 조절할 수 있다. 본 연구에서는 이러한 지시자의 부적절한 사용이 미칠 수 있는 영향에 대해 분석한다.

본 논문에서는 위와 같은 세 가지 상황에 대한 실험을 구체적으로 다음과 같은 방법으로 진행한다. 첫 번째는 동시 접속자의 수가 급증하는 상황에서 응답 시간 및 처리율의 변화를 분석한다. 두 번째는 서버의 계산 시간의 변화에 따른 사용자 응답 시간 및 처리율을 측정 및 분석한다. 마지막으로, 아파치 웹 서버 지시자의 변화에 따른 아파치 2.0 웹 서버의 성능 변화를 분석한다.

3.2 시뮬레이터

실험을 진행하기 위하여 본 논문에서는 아파치 웹 서버와 연동할 수 있는 시뮬레이터를 자체적으로 제작하였다. 시뮬레이터는 리눅스 환경 하에서 C언어를 이용하여 작성하였다. 그림 2는 클라이언트 시뮬레이터의 전체적인 구성을 보여준다.

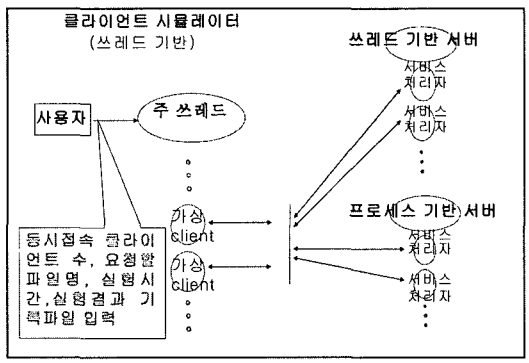

그림 2 시뮬레이터 구성

프로그램이 실행되면 주 쓰레드는 입력한 동시접속 클라이언트 수만큼 쓰레드들을 생성한다. 이 쓰레드들이 클라이언트를 묘사하는 가상 클라이언트가 된다. 생성된 가상 클라이언트들은 입력한 동시접속 클라이언트 수만큼의 클라이언트가 모두 생성될 때까지 pthread_ cond_ wait() 함수를 통해서 기다린다. 그 후 모든 가상 클라이언트들이 생성 완료가 되면 pthread_cond_broadcast()를 통해서 동시에 접속을 시도하게 된다. 만약 서버가 바쁜 이유로 시간만료(ETIMEDOUT)가 되어서 접속에 실패하게 된다면 접속을 다시 시도하게 된다. 접속에 성공하면 사용자가 입력한 요청파일을 웹 서버에게 요청한다. 접속된 후 사용자가 입력한 파일을 요청하고 나면 응답시간을 기록하기 위한 첫 번째 시간을 측정한다. 그리고 나서 웹 서버로부터 응답을 받게 되면 다시 한번 시간을 측정해서 응답 받은 후의 시간과 요청을 보낸 직후의 시간의 차이를 실험 결과 기록파일에 기록한다. 시간 측정을 위해서 gettimeofday() 함수를 사용한다. 생성된 가상 클라이언트들은 반복적으로 웹 서버에게 사용자가 입력한 파일을 요청하기 위해 접속을 시도한다.

모든 가상 클라이언트들이 이와 같은 방법으로 사용자가 입력한 실험 시간 동안 응답시간을 측정한 후에 프로그램은 종료하게 된다. 프로그램이 종료한 후 생성

된 실험결과 기록파일은 간단한 스크립트를 통해서 처리율과 평균 응답시간을 알아볼 수 있다.

3.3 실험 환경

실험은 256 MB의 메모리와 Pentium III 1.0 GHz의 CPU를 탑재한 레드햇 8.0 리눅스에서 진행한다. 실험을 위하여 두 대의 서로 다른 컴퓨터에서 서버와 클라이언트를 설정한다. 두 대 모두 동일한 사양이다. 실험에서는 앞서 기술한 바와 같이 아파치 2.0 웹 서버를 사용하였으며, 3.2 절에서 제시한 시뮬레이터를 연동하여 실험한다.

4. 실험 및 결과 분석

이 절에서는 본 연구를 통해 이루어진 세 가지 실험에 대한 실험 결과 및 분석 내용을 정리한다.

4.1 사용자 폭주로 인한 동시 접속자 수 증가 실험

프로세스 서버와 쓰레드 서버의 성능을 측정하기 위해서 두 서버 모두에게 일정한 양의 작업을 부여한다. 작업은 약 0.2초의 시간이 걸리는 계산 작업이고 클라이언트로부터 요청이 들어왔을 때 이 작업을 수행하고 응답을 보내주게 된다. 표 2는 실험에 사용 된 변수와 그 외의 설정 내용을 보여준다.

표 2 동시접속 클라이언트의 변화에 따른 서버의 성능 측정을 위한 실험 설정

실험시간	2분
동시접속 클라이언트 수의 변화	10~100개
서버에서의 작업량(계산시간)	0.2초

두 서버의 실험 결과는 프로세스를 이용했을 때 보다 쓰레드를 이용했을 때 문맥전환에 걸리는 오버헤드가 더 적기 때문에 쓰레드를 이용한 서버의 응답시간과 처리율 측면에서 더 우수할 것으로 예상된다. 그림 3과 4의 실험결과는 예상과 마찬가지로 문맥전환에서의 오버헤드가 더 적은 쓰레드 서버가 응답시간과 처리율 면에서 우수함을 보여준다. 그리고 두 서버의 응답시간의 차이가 거의 일정하게 나타났고 처리율도 크게 다르지 않은 것으로 볼 때 클라이언트의 변화는 두 서버의 성능

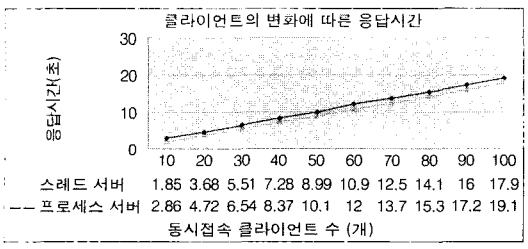

그림 3 동시접속 클라이언트수의 변화에 따른 응답시간

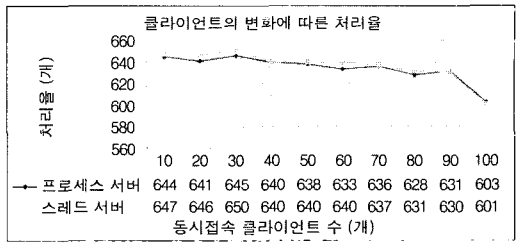

그림 4 동시접속 클라이언트수의 변화에 따른 처리율

에 비슷한 작용을 하고 있다는 것을 알 수 있다. 이것은 프로세스 서버와 쓰레드 서버에 대해서 클라이언트가 증가하는 것에 대한 오버헤드는 일관되게 작용한다는 것을 의미한다.

4.2 서버 계산 시간의 증가에 따른 영향력 분석 실험

이 실험은 첨단과학 기술의 문제를 웹을 이용하여 해결하고자 하였을 때 프로세스 서버와 쓰레드 서버가 어떠한 영향을 받는지를 알아보기 위한 실험이다. 이번 실험에서는 클라이언트 수를 고정시키고 서버에서 프로세서를 사용하는 계산 작업 시간을 점점 증가하였을 때 응답시간과 처리율의 변화를 살펴보았다. 표 3은 실험에 사용된 환경 설정 내용을 보여준다.

표 3 CPU 점유시간의 변화에 따른 서버의 성능 측정을 위한 실험 환경 설정

실험 시간	20분
요청이 하나일 때 서버의 계산시간의 변화	2초~10초 (2초 간격)
동시접속 클라이언트 수	50개

그림 5는 CPU 점유시간의 변화에 따른 두 서버의 응답시간을 측정한 결과이다. 쓰레드 서버가 프로세스 서버보다 빠른 응답시간을 보여준다. 표 4는 두 서버의 처리율을 나타낸다. 이 두 서버의 처리율은 서버에서 작업하는 시간이 길기 때문에 서로 다르지 않게 나타나는 것으로 분석된다.

그림 6은 클라이언트 수가 50개 일 때 응답시간에서 네트워크에 걸리는 시간을 제외한 서버에서의 처리시간

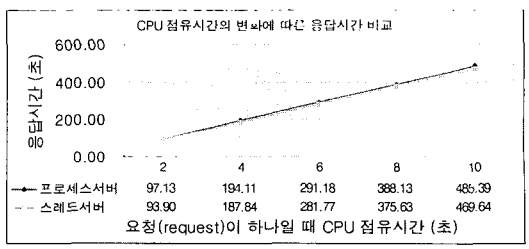

그림 5 CPU 점유시간의 변화에 따른 응답시간

표 4 CPU 점유시간의 변화에 따른 두 서버의 처리율

CPU 점유시간(초)	2	4	6	8	10
프로세스 서버(개)	600	300	200	150	100
쓰레드 서버(개)	600	300	200	150	100

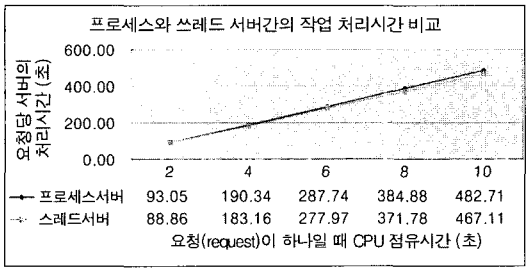

그림 6 프로세스와 쓰레드 서버의 요청에 대한 처리시간 비교

(문맥전환 오버헤드 포함)만을 측정한 것이다. 그림 5와 6에서 요청이 하나일 때 CPU 점유시간의 변화에 대해서 프로세스와 쓰레드 서버의 응답시간과 처리시간의 차이는 점점 증가하였고, 처리시간만을 측정하였을 때가 더 큰 차이를 보여주고 있다. 이것은 서버에서 프로세서를 사용하는 계산 작업의 양이 많으면 많을수록 쓰레드 서버와 프로세스 서버의 성능의 차이는 커진다는 것을 의미한다.

4.3 아파치 지시자 변화에 대한 실험

이 실험을 위하여 아파치의 지시자들을 다음과 같은 두 가지 유형으로 설정한다. 하나의 설정은 계속되는 사용자의 요구를 처리하기 위해서 멀티 프로세스를 주로 활용하도록 하고, 다른 하나는 멀티 쓰레드를 주로 활용하도록 하는 것이다. 구체적으로 설명하자면 다른 지시자는 기본 설정에서 변경하지 않은 상태에서 시작하는 프로세스 개수인 StartServers와 서버 프로세스의 개수를 제한하는 ServerLimit 그리고 ThreadPerChild를 변화하였다. 쓰레드를 주로 사용하는 아파치와 프로세스를 주로 사용하는 아파치의 처음 시작하는 쓰레드의 개수(StartServers*ThreadPerChild)는 같으며, 동시에 처리할 수 있는 최대 클라이언트 수(MaxClients)도 같게 설정하였다. 표 5에서 그 두 가지 설정을 자세히 보여준다.

쓰레드의 문맥전환의 오버헤드는 프로세스의 70% 정도라고 알려져 있기 때문에[4], 쓰레드를 주로 사용하는 경우에 응답시간과 처리율 면에서 우수할 것으로 예상된다. 그러나 실제 실험 결과에 의하면 예상과는 다르게 쓰레드를 주로 사용하는 설정을 가진 아파치가 프로세스를 주로 사용하는 아파치 보다 더 좋지 못한 결과를 보여준다. 그림 7과 8에서 볼 수 있듯이 응답시간은 프로세스를 주로 사용하는 아파치가 더 빠르게 나왔고, 처

표 5 프로세스와 쓰레드 비교 실험을 위한 아파치 지시자 설정

지시자(directive)	쓰레드를 주로 사용하는 경우	프로세스를 주로 사용하는 경우
ServerLimit	3	150
StartServers	1	50
MaxClients	150	150
MinSpareThreads	25	25
MaxSpareThreads	75	75
ThreadsPerChild	50	1
MaxRequestsPerChild	0	0

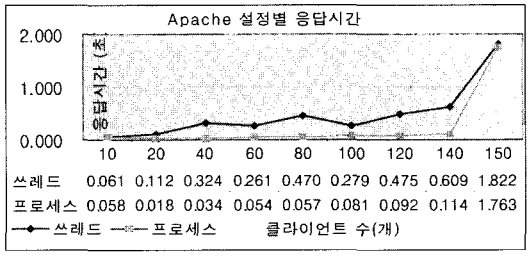

그림 7 표 5의 경우 쓰레드를 주로 사용하는 아파치와 프로세스를 주로 사용하는 아파치의 응답시간 비교

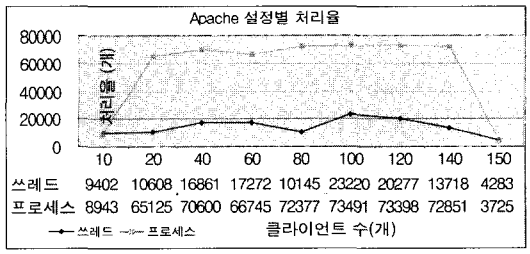

그림 8 표 5의 경우 쓰레드를 주로 사용하는 아파치와 프로세스를 주로 사용하는 아파치의 처리율 비교

리율 면에서도 프로세스를 주로 사용하는 경우의 처리율이 더 많은 것으로 나타났다. 이것은 멀티 프로세스와 쓰레드를 사용하는 과정에서 다른 요인이 작용한 것으로 판단된다.

이와 같은 결과에 대한 원인을 분석하기 위하여 다음과 같은 추가 실험을 진행하였다. 표 6은 추가 실험에 사용된 지시자의 내용을 보여준다. 앞선 실험에서는 쓰레드들을 미리 생성해 놓고 스페어 쓰레드들을 정해진 지시자에 맞는 설정을 유지하기 위해서 지속적으로 쓰레드를 가진 프로세스를 생성하고 제거하는 작업을 반복한다. 그러나 표 6을 통한 실험에서는 스페어 쓰레드들을 유지하기 위해서 자식 프로세스를 제거하거나 생성하는 일이 없다. 처음 시작 시 총 150개의 쓰레드들을

생성해 놓으면 그 쓰레드의 개수는 MinSpareThreads와 MaxSpareThreads인 150개에 적절하다. 따라서 요청이 들어왔을 때 스페어 쓰레드들이 줄어들게 되어도 ServerLimit에서 제한을 받기 때문에 프로세스를 추가로 생성하지 않는다.

그림 9와 10은 추가 실험에 대한 실험 결과이다. 실험의 결과는 예상대로 쓰레드를 주로 사용한 아파치가 응답시간과 처리율 면에서 우수한 성능을 보여준다. 이상과 같은 실험 결과를 볼 때, 첫 번째 실험에서 쓰레드 위주의 아파치가 나쁜 성능을 보여준 것은 스페어 쓰레드를 유지하는데 드는 비용이 쓰레드를 이용한 장점을 살리지 못한 주요 원인으로 분석된다. 이것은 다중 쓰레드를 웹 서버에 이용하는 데 있어서 성능을 저하시킬

표 6 오버헤드 분석 실험을 위한 아파치 지시자 설정

지시자(directive)	쓰레드를 주로 사용하는 경우	프로세스를 주로 사용하는 경우
ServerLimit	3	150
StartServers	3	150
MaxClients	150	150
MinSpareThreads	150	150
MaxSpareThreads	150	150
ThreadsPerChild	50	1
MaxRequestsPerChild	0	0

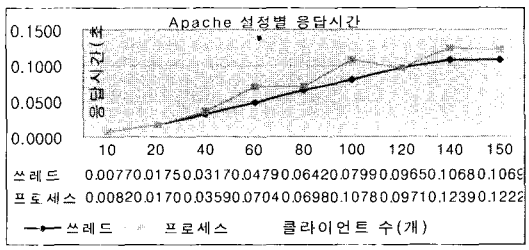

그림 9 표 6의 경우 쓰레드를 주로 사용하는 아파치와 프로세스를 주로 사용하는 아파치의 응답시간 비교

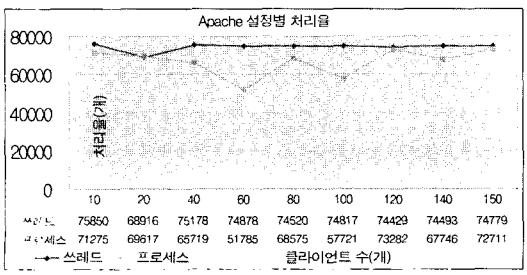

그림 10 표 6의 경우 쓰레드를 주로 사용하는 아파치와 프로세스를 주로 사용하는 아파치의 처리율 비교

수 있는 여러 요소들을 고려해야만 프로세스를 이용한 웹 서버보다 더 좋은 성능을 가질 수 있다는 것이다. 그 중 응답시간을 단축하기 위해서 고안된 스페어 쓰레드들의 범위에 대한 더 많은 고찰이 필요할 것이다.

5. 결론 및 향후 연구 과제

본 논문에서는 멀티 프로세스와 멀티 쓰레드를 웹 서버에 이용할 경우에 있어서 영향을 끼칠 수 있는 요인들에 대해 실험적인 방법을 통해 분석하였다. 실험 결과에 의하면, 차세대 컴퓨팅 환경, 즉, 사용자가 폭주하여 동시접속자의 수가 증가하거나, 바이오 애플리케이션이나 그리드 컴퓨팅과 같은 서버의 계산 시간을 많이 요구하게 되는 환경은 다중 쓰레드의 사용에 별다른 영향을 미치지 않음을 알 수 있었다.

오히려 멀티 프로세스와 멀티 쓰레드를 혼용하기 위해 사용되는 다양한 형태의 웹 서버 지시자가 쓰레드의 효율성에 좋지 않은 영향을 미칠 수 있음을 실험을 통해 확인하였다. 점점 증가하는 웹 사용자들에 효율적으로 대처하기 위해서 웹 서버는 더욱 강력하고 안정된 성능을 요구받는다. 이러한 요구에 대응하기 위해서 웹 서버는 더 높은 확장가능성과 함께 변화하는 사용자 환경에 대해 능동적으로 최적의 서비스를 제공할 수 있도록 개선되어야 할 것이다.

현재 우리는 가장 좋은 성능을 낼 수 있는 웹 서버의 멀티 쓰레드와 멀티 프로세스의 구조를 자동으로 설정하기 위한 연구를 진행하고 있다. 진행되는 연구를 통해 다양한 상황에 맞는 적절한 설정이 어떤 것인지에 대한 방향을 제시하게 될 것이다.

참 고 문 헌

[1] The Apache Software Foundation. http://www.apache.org, 1999-2003.

[2] Ryan B. Bloom, Apache Server 2.0: The Complete Reference, McGraw-Hill Osborne Media, 1st edition (June 26, 2002).

[3] Mi-ryeong Yeom, Kihun Chong, Sam H. Noh, An Assessment of the Apache Web Server 2.0 Performance on Linux, In Proceedings of 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2002) July 14-18, 2002.

[4] http://www.atnf.csiro.au/~rgooch/benchmarks/linuxscheduler.html

전 홍 석

1996년 홍익대학교 컴퓨터공학과(학사) 1998년 홍익대학교 대학원 전산학과(석사). 2001년 홍익대학교 대학원 전산학과(박사). 1997년~1999년 (주)팅크웨어시스템즈 선임연구원. 2000년~2001년 (주)이칼로스 기술담당 부사장. 2001년~2002년 (주)이씨앤아이티 기술연구소장. 2002년~현재 건국대학교 컴퓨터응용과학부 조교수. 관심분야는 시스템소프트웨어, 지능형로봇, 모바일컴퓨팅

이 승 원

2004년 건국대학교 자연과학대학 컴퓨터과학과(학사). 2004년~현재 건국대학교 컴퓨터정보통신공학부(석사과정). 관심분야는 임베디드 시스템, GIS, LBS

강 현 규

1985년 홍익대학교 전자계산학과(학사) 1987년 한국과학기술원 전산학과(석사) 1992년 정보처리 기술사 자격 취득. 1997년 한국과학기술원 전산학과(박사). 1987년~2000년 한국전자통신연구원 책임연구원. 2001년~현재 건국대학교 컴퓨터응용과학부 조교수. 관심분야는 정보검색, 자연어처리, 한국어 정보처리, XML, CRM, WML 등