다중 처리기 기반 웹 서버 구조의 실험적 성능 분석

(Experimental Analysis of Web Server on Multiprocessor)

정 진 국 † 낭 종 호 †† 박 성 용 †††

(JinGuk Jeong)(JongHo Nang)(SungYong Park)

요 약 WWW의 급격한 발전은 몇몇 웹 서버에 큰 과부하를 가져오게 하였다. 이로 인해 고성능 웹 서버의 구축이 필요하게 되었는데, 그 중 프로세스의 오버헤드를 줄이기 위해 도입된 멀티 쓰레드 기법을 이용한 병행 웹 서버들이 많이 이용되게 되었다. 일반적으로 멀티 쓰레드 기법을 이용하는 웹 서버의 구조는 요구 기반 웹 서버, 작업 기반 웹 서버, Thread Pool 구조 웹 서버 등으로 나눌 수 있다. 본 논문에서는 이런 웹 서버들을 리눅스가 탑재되어 있는 다중 처리기상에서 구현하였으며, 다양한 환경하에서 성능을 비교, 분석하였다. 각각의 웹 서버들은 Pthread 라이브러리와 Socket 라이브러리를 이용하여 구현하였으며, 여러 파라미터 - CPU 개수, CGI 비율, 웹 서버 구조, 파일 크기, 부하량 등 - 를 조절하면서 실험하였다. 실험 결과 분석에 의하면 요구 기반 웹 서버에서는 하나의 CPU에서 수행이 되는 쓰레드의 개수가 많게 되면 성능이 저하되는 것을 볼 수 있었고, 작업 기반 웹 서버에서는 단계 사이의 불균형으로 인하여 큐에서의 대기 시간이 누적되면 성능이 저하되는 것을 볼 수 있었다. Thread Pool 구조의 웹 서버는 쓰레드의 개수가 조절되고, 큐에서의 대기 시간 또한 없앰으로 해서 다른 웹 서버에 비해 좋은 성능을 보임을 알 수 있었다. 이와 같은 실험 결과는 다중 처리기를 이용한 고성능 웹 서버를 구축하는 데 있어서 이용될 수 있을 것이다.

Abstract The explosive growth of the WWW caused an overload on several famous Web server, and this fact stimulated a research to develop a high performance concurrent Web servers based the multithreaded architecture. Generally, concurrent Web servers could be classified into RBW(Request Based Web server), TBW(Task Based Web server), and TPW(Thread Pool Web server) with respect to the way to incorporate the multithreaded into Web architecture. In this thesis, these three Web servers are implemented with pthread library and socket library on 4-CPUs multiprocessor under LINUX, and their performance are analyzed experimentally while changing the parameters such as the number of CPU, the requested file size, amount computations for CGI, and the request rates.

From the experiments, we could find out that the performance of RBW is degraded when there are more threads than the system can handle, and that of TBW is degraded when the waiting time in ready queue is accumulated because of the unbalance between the subtasks. On the other hands, the TPW could produce a better performance regardless of the parameters because the number of threads in TPW is fixed so that an overloaded situation is never occurred and there is no waiting time in TPW. These experimental analyses could be used as a reference model to build a high performance Web server on multiprocessors.

† 비 회 원 : 서강대학교 컴퓨터학과
jjguk@mlneptune.sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
jhnang@ccs.sogang.ac.kr
††† 비 회 원 : 서강대학교 컴퓨터학과 교수
parksy@ccs.sogang.ac.kr
논문접수 : 2000년 4월 21일
심사완료 : 2000년 12월 15일

1. 서 론

월드 와이드 웹(World Wide Web;이하 웹)은, 불과 몇 년 사이에 폭발적인 사용자들의 증가로 인해서 이제는 국가적인 권장을 하고 있는 추세를 보이고 있기까지 하다. 현재의 웹은 VOD, 전자 상거래와 같이 사용하고 있는 분야는 물론이고, 서로 주고 받는 데이터의 형식도

단순한 텍스트 형식에서 멀티미디어 데이터를 주고 받는 등 다양해지고 있다. 이와 같은 급격한 웹의 발전은 몇몇 웹 서버에 사용자의 요구가 집중되는 현상을 보이게 하였다. 특히 서버에서 동적으로 수행되는 프로그램들 - CGI(Common Gatewat Interface), Script, Java Servlet - 을 사용하는 빈도가 많아지고 있으므로 사용자의 요구가 집중되는 현상은 몇몇 서버에 큰 과부하를 가져오게 할 것이다. 서버에 과부하가 생기면 데이터를 주고 받는 데 시간이 오래 걸리게 된다. 그러므로 사용자의 요구에 신속하게 응답할 수 있는 고성능 웹 서버의 구축이 필요하게 된다.

네트웍의 성능이 점점 고속화 됨에 따라 웹 서버 자체의 성능이 전체 웹 구조의 성능에 큰 영향을 끼치게 되었다. 그로 인해 웹 서버 자체의 성능 향상을 위해 많은 연구가 이루어져 왔고, 현재도 이루어지고 있는 상태이다. 웹 서버 성능 향상을 위해서 제안되고 있는 방법들은 입출력의 속도를 향상시키기 위한 캐슁 기법[5]과, 웹 상에서의 요구 분석을 통해 응답 순서를 스케쥴링하는 SRPT(Shortest Remaining Processing Time) 스케쥴링 기법[3] 등을 들 수 있다. 또한 순차 수행 웹 서버의 한계와 프로세스 생성 시간에 의한 많은 부하로 인하여 다양한 웹 서버 구조들이 나타나게 되었는데 프로세스 풀[4]을 이용하는 방법, 쓰레드를 이용하여 병행성을 추구하는 방법[9,10] 등이 이로 인해 나타난 웹 서버 구조들이다. 실제로 이러한 기법들을 통하여 어느 정도의 성능 향상을 가져올 수 있음은 실험을 통하여 입증이 되어 있다. 하지만 이러한 연구들은 단순히 텍스트와 그림 파일과 같은 작은 파일들의 요구에 기반을 두고 있기 때문에 동적인 프로그램의 요구나 멀티미디어 파일과 같은 큰 파일들의 요구가 늘어나는 최근의 웹 상의 전송방식에 적용하는 데는 문제점이 발생하게 되고, 또한 기존의 연구들은 단일 처리기상에서의 실험에 중점을 두고 있어 그 결과가 최근 많이 보급되어 있는 다중 처리기에서도 보장되는 지는 알 수 없다는 문제점이 있다.

본 논문에서는 최근 널리 보급되고 있는 다중 처리기 상에서 어떤 병행 웹 서버 구조가 고성능을 낼 수 있는 지에 대해 실험을 통하여 분석하였다. 이를 위하여 멀티 쓰레드 기법을 이용한 대표적인 3종류의 병행 웹 서버를 리눅스가 탑재되어 있는 다중 처리기 상에서 구현하였으며, 다양한 요구들(정적 파일 요청, 동적 프로그램 요청)과 환경(파일 크기, CPU 개수, 동적 프로그램 요구의 비율)에서 성능을 측정하였다. 본 논문에서 고려한 병행 웹 서버 구조는 세 가지로 요구 기반 병행 구조(Request Based Web server), 작업 기반 병행 구조(Task Based Web server), Thread Pool 구조(Thread Pool Web server) 이다. 요구 기반 병행 구조는 병행처리 단위가 클라이언트의 요구이고, 작업 기반 병행 구조는 서버측의 처리 단계이다. 그리고 Thread Pool 구조는 두 구조를 혼합한 형태의 구조이다.

실험을 위한 프로그램은 리눅스 운영체제 위에서 pthread 라이브러리[15]와 socket 라이브러리[16]를 이용하여 구현되었다. 리눅스 운영체제는 각각의 쓰레드를 각각의 CPU로 할당해주는 역할을 운영체제 상에서 제공을 해주기 때문에 실제로 쓰레드들이 병렬로 수행이 될 수 있다. 본 논문에서는 웹 서버의 수행 과정에 내재되어 있는 병행성을 찾아 여러 환경상에서 실험을 하였다. 실험 결과를 통해 클라이언트의 요구가 많아질수록 서버의 처리율이 낮아짐을 볼 수 있었고 동적인 프로그램에 대한 요구가 많아질수록 다중 처리기 상에서의 웹 서버가 단일 처리기 상에서의 웹 서버 보다는 많은 처리율을 보임을 알 수 있었다. 또한 본 논문의 실험 환경에서는 Thread Pool 구조가 전체적으로 좋은 성능을 보임을 알 수 있었다. 이와 같은 결과는 고성능 웹 서버를 구축하고자 할 때 각각의 환경에 적합한 구조를 찾는 데 참고 모델로 사용될 수 있을 것이다.

2. 연구 배경

본 장에서는 다중 처리기에서의 웹 서버 성능 평가를 위해서 필요한 웹 서버의 기본 구조와 선행되었던 웹 서버에 대한 연구 내용을 살펴보도록 하겠다. 2.1절에서는 순차적으로 처리가 되는 웹의 기본 구조에 대해 간단한 설명을 하며, 2.2절에서는 기존에 선행된 연구 내용에 대한 설명을 하도록 한다.

2.1 웹의 기본 구조

웹 서버의 기본 알고리즘[6,7]은 TCP/IP 상에서 연결되어진 클라이언트를 통해 받은 요구에 대해 URI을 구하고 그에 해당하는 문서를 HTTP 프로토콜에 맞추어서 보내주는 것이다. HTTP/1.1의 지속적인 연결에 대한 항목을 이용하면 하나의 TCP 연결에서 위의 과정을 되풀이하게 되지만 본 논문에서는 HTTP/1.0에 맞추어 구현을 하였으므로 웹 서버의 기본 구조 또한 HTTP/1.0의 특성에 맞추어 설명하도록 하겠다.

<그림 1>은 웹 서버의 기본 알고리즘에 관한 그림이다. 웹 서버는 TCP 연결을 통해 클라이언트로부터 요구를 받아들인 후에 요구를 읽고 그 요구에서 필요한정보를 얻게 된다.

분석하는 데에는 클라이언트와 서버 사이에 미리 정해진 규칙들이 적용이 되는 데 그것이 바로 HTTP 프

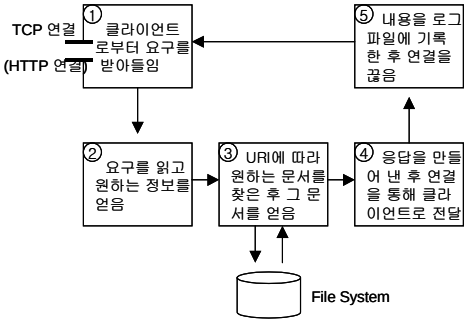


그림 1 웹 서버의 기본 알고리즘

로토콜이다. 요구 분석 후에 서버는 URI에 따라 클라이언트가 원하는 문서를 찾게 된다. 문서를 찾은 후 서버는 요구에 맞는 응답을 만들어내고, 그 응답을 클라이언트에게 보내게 된다. 그 후 로그 파일을 기록하고 연결을 끊으면 한 요구에 대한 모든 과정이 끝나게 되는 것이다.

웹 서버는 요구의 메쏘드에 따라서 서로 다른 행동을 보인다. 하지만 보통 사용되는 메쏘드는 Get이 대부분이고, Delete나 Put과 같이 서버의 내용을 변화시키는 메쏘드는 거의 모든 웹 서버에서 제공을 하지 않는다.

그리고 Post와 같은 메쏘드는 CGI와 같은 서버에서 수행되는 프로그램에 입력으로 들어가는 데이터를 받기 위해서만 이용되지 다른 용도에서는 제공을 하지 않는 것이 일반적이다. 그러므로 현재의 웹 서버라는 것은 사용자가 원하는 문서나 혹은 프로그램의 결과를 보내주는 것으로 이해하면 될 것이다. 또한 본 논문에서 실험을 위해 이용되는 데이터 또한 이러한 이유로 Get 메쏘드에 초점을 두어 실험을 하도록 할 것이다.

2.2 웹 서버에 관한 기존의 연구들

기존의 웹 서버에 관한 연구들은 세 가지 정도의 영역으로 이루어져 왔다. 첫 번째는 웹 서버 자체의 성능 향상에 중점을 둔 연구[3,5,10], 두 번째로는 요구에 대한 웹 서버의 시간 측정을 하여 각각의 요소별로 어느 정도 시간이 걸리는 지를 알아보는 연구[1], 마지막 세 번째로는 웹 서버로 입력되는 요구들의 종류를 분류한 뒤 각각의 문서에 대한 접근의 비율은 어느 정도이고, 문서의 종류에 따라 어떤 특징이 있는 지를 알아보는 연구[8]이다. 본 논문에서 언급되는 내용이 웹 서버의 성능 향상에 중점을 두는 것이 아니므로 여기서는 두 번째와 세 번째의 연구 종류에 대해서 설명하도록 하겠다. 우선 2.2.1절에서는 정적 파일의 요구에 대한 웹 서버의 시간 측정과 분석에 관한 기존의 연구를 설명하도록 하겠고 2.2.2절에서는 CGI를 중심으로 한 요구에 대해 웹 서버의 성능은 어떻게 측정되는 지에 관한 기존의 연구를 설명하도록 하겠다.

2.2.1 정적 파일의 요구에 대한 웹 서버의 시간 분석

Rhode Island Kingston 대학에서는 Apache 웹 서버를 이용해 요구를 처리하는 데 걸리는 시간에 대한 분석[1]을 하였다. 파일 크기에 따라 Class 0에서 Class 3까지 나누어서 실험을 하였는데, 파일의 크기가 작을수록 사용자 공간에서 걸리는 시간의 비율이 크며 파일의 크기가 클수록 네트웍에서 걸리는 시간의 비율이 큼을 알 수 있다. 사용자 공간에서 걸리는 시간에는 요구를 분석하여 원하는 정보를 얻어내는 데 걸리는 시간이 대부분을 차지한다. 그리고 네트웍에 걸리는 시간에는 TCP/IP에서 걸리는 시간과 Ethernet 드라이버와 같이 Low level의 네트웍 관리에 드는 시간들이 포함이 된다. 실제로 네트웍에 걸리는 시간 중 대부분이 되는 것은 TCP/IP 프로토콜에 필요한 시간이다. 특히 TCP 연결에서는 연결을 해제할 때 새로운 연결이 과거의 데이터를 받지 못하도록 어느 정도의 시간을 기다리게 되는데 그 때의 상태를 TIME_WAIT라고 한다. 이 때 걸리는 시간이 TCP를 연결할 때마다 발생하기 때문에 큰 비율을 차지하게 된다.

Rhode Island Kingston 대학 연구[1]는 웹 서버 상에서의 요구 처리 시간을 분석한 후 Apache 웹 서버의 성능 향상을 위한 몇 가지 제안을 하였다. 그 제안의 대부분은 캐슁 기법에 관한 내용이다. 문서를 사용자 공간에 캐슁을 하는 기법, DNS(Domain Name Service)의 결과를 저장하는 기법, 로그 파일에 기록할 내용을 미리 캐슁하는 기법, 요구를 읽고 분석하는 데 있어 거의 대부분의 요구는 그 전 클라이언트에서 온다는 것에 기본을 두어 요구 분석 결과의 어느 정도를 캐슁하는 기법 등이 있으며, 이외에 문서를 읽어오는 데 있어 mmap 함수를 이용하는 방법을 제시했는데 이 함수를 이용하는 경우에는 성능이 많이 향상되지는 않았다. 실제 이런 방법 모두를 적용한 실험에서 결과는 50%이상 좋아짐을 알 수 있었다. 하지만 이러한 방법에는 문제점이 발생할 수 있다. 이들이 실험에서 사용한 문서 중 가장 큰 문서는 1Mbytes의 문서이다. 하지만 요즘에 이용되는 문서들의 크기는 이보다 훨씬 큰 것들이 많다. 큰 파일들에도 똑같은 식으로 캐슁을 한다는 것은 메모리가 아주 크지 않고는 문제가 생길 것이다. 또한 메모리의 크기를 늘린다는 것 또한 한계가 있기 때문에 해결책이 되지는 않는다. 그러므로 무작정 캐슁을 해서 데이터를 저장하는 방법은 무리가 따르게 된다.

Boston 대학의 연구[2]는 문서의 크기에 따라 웹 서

버의 Throughput이 어떻게 바뀌는 지를 알아보기 위한 것이다. 처리량을 두 가지로 나눴는데, 하나는 초당 처리할 수 있는 클라이언트의 수이고, 다른 하나는 초당 처리할 수 있는 문서의 총 크기이다. 결과를 보면 초당 처리할 수 있는 클라이언트의 수는 문서의 크기가 크면 클수록 작아지지만 초당 처리할 수 있는 문서의 총 크기는 점점 커짐을 볼 수 있다. 이것은 클라이언트에서 원하는 문서의 크기가 클수록 서버가 서비스할 수 있는 클라이언트의 수는 작아짐을 보이는 것이다. 이것은 한 클라이언트를 처리하는 시간이 점점 늘어나기 때문이다. 하지만 한 클라이언트를 위해 처리하는 문서의 크기가 커지기 때문에 초당 처리할 수 있는 바이트 수는 커지는 것이다.

보통 정적 파일의 요구에 대한 시간 분석을 하는 기존의 연구들은 거의 위와 비슷한 내용을 갖는다. 환경에 따라 조금씩 달라지기는 하겠지만 달라지는 폭이 그리 크게 나타나지는 않는다. 대부분 하나의 요구를 처리하는 데 있어서 네트웍이 가장 큰 시간을 차지하고, CPU에서 걸리는 시간은 크게 나타나지 않는다. 하지만 이러한 연구 중 대부분이 CPU 개수와 웹 서버 구조는 고려하지 않은 연구들이라서 다중 처리기 혹은 다른 웹 서버 구조에 그대로 적용할 수 있을 지는 알 수 없다.

2.2.2 CGI 요구에 대한 웹 서버의 성능 측정

Watson 리서치 센터에서는 CGI와 같은 동적인 프로그램에 대한 요구가 웹 서버의 성능에 어떻게 영향을 주는 지 실험[13]을 하였다. 여기서 파라미터로 이용한 것은 CV라는 큐의 길이에 관한 것이다. 즉, 이들은 웹 서버에 요구를 받아들이는 큐를 이용했는데, CV가 1이라는 것은 큐를 이용하지 않았을 때, 즉, 큐의 길이가 0일 때를 나타내고, CV가 2라는 것은 큐의 길이가 24라는 것이다. CV가 3이면 큐의 길이는 66이 되고, CV가 4라는 것은 120 크기의 큐를 나타낸다. 결과를 보면 두 가지 사실을 알 수 있다. 하나는 CGI의 비율을 나타내는 X축의 값이 늘어날수록 그에 따라 Latency도 늘어난다는 사실이고, 다른 하나는 CV의 값이 늘어날수록 Latency도 늘어난다는 사실이다. 전자는 CGI의 비율이 늘어날수록 그에 따라 CPU의 부하가 늘어나게 되므로 전체 웹 서버의 성능도 저하된다는 사실을 말해준다. 그리고 후자는 큐의 값이 늘어나면 늘어날수록 그만큼 큐에서 기다리는 시간이 첨가가 되기 때문에 Latency도 증가한다는 사실을 말해주는 것이다.

Watson 리서치 센터의 연구[12]의 실험 결과에서 설명하는 내용은 CGI의 요구를 받아들이는 경우에 CPU 처리 능력이 전체 웹 서버의 성능을 좌우할 수 있다는 것이다. CGI에 대한 요구를 처리하기 위해서는 fork( ), exec( ) 와 같은 과정을 거쳐서 프로세스를 생성해야 하고 웹 서버는 그 프로세스가 수행을 종료할 때까지 기다려야 한다. 이런 과정은 다른 과정에 비해 CPU를 많이 사용하게 되어 CGI에 대한 요구를 처리하는 것이 다른 요구를 처리하는 것에 비해 CPU를 많이 사용하게 되는 것이다. 정적 파일에 대한 요구를 처리하는 경우에는 CPU 처리에 관한 부분이 거의 없기 때문에 네트웍만 따라준다면 1초당 수백개의 요구도 처리가 가능하지만 CGI와 같은 동적 프로그램의 요구를 처리하는 경우에는 네트웍이 따라준다 하더라도 CPU 처리 능력이 웹 서버 성능의 제한 요소가 되어 전체 능력을 떨어트리는 것이다. 또한 부가적으로 실험에 의해서 나타나는 내용은 큐와 Latency 그리고 웹 서버 처리 능력의 상관관계에 관한 내용이다. 큐의 크기가 클 때는 전체 요구 중 에러 즉, 웹 서버가 받아들이지 않는 요구는 줄어들지만 클라이언트가 응답을 받는 데까지 걸리는 시간은 길어지고, 큐의 크기가 작다면 클라이언트가 응답을 받는 데까지 걸리는 시간은 줄어들지만 웹 서버가 받아들이지 않아서 에러가 나는 요구의 수는 많아진다는 것이다. 이런 상호간의 관계가 있기 때문에 큐의 크기는 환경에 따라 조절할 수 있을 것이다.

Watson 리서치 센터의 연구 결과[12]에 의해 CGI를 포함하는 동적 프로그램의 요구에 대한 처리에서는 CPU 처리 능력이 중요하다는 것이 증명 되었다. 하지만 위의 연구는 CPU 개수의 변화에 대한 실험이 없어 요즘 많이 보급되는 다중 처리기에서 그 실험 결과를 그대로 적용할 수 있을 지 보장할 수 없고, CGI 비율을 통해서만 증명이 되어 있으므로 다른 파라미터에 의해 어떤 변화가 발생할 수 있는 지 또한 알 수 없다. 그리고 여러 웹 서버 구조에 대한 비교가 없으므로 실험에 적용된 웹 서버 구조를 제외한 다른 구조에서는 어떻게 CGI에 대한 성능이 나타날 지도 알 수 없다.

3. 병행 웹 서버의 구현

기존 연구들의 대부분은 하나의 웹 서버 구조를 가지고 실험을 하였다. 그로 인해 다른 웹 서버 구조에서는 어떻게 실험 결과가 나올 지 알 수 없었고, 웹 서버 구조 사이에서는 어떻게 성능 차이가 나는 지 알 수 없었다. 그렇기 때문에 본 논문에서는 현재 많이 쓰이고 있는 세 가지 웹 서버 구조를 이용하여 실험을 하였다. 본 논문에서 이용한 세 가지 웹 서버 구조는 다음과 같다. 클라이언트로부터 요구가 들어오면 쓰레드를 생성하는 요구 기반 웹 서버(Request Based Web server; 이하

RBW)[10], 순차 수행 웹 서버의 각각의 단계를 중심으로 각각의 단계마다 쓰레드를 생성하는 작업 기반 웹 서버(Task Based Web server; 이하 TBW)[14], 그리고 마지막으로 이 두 가지 웹 서버 구조를 혼합 시킨 형태의 웹 서버인 Thread Pool 구조 웹 서버(Thread Pool Web server; 이하 TPW)[11,13]등이 그것이다. 이 장에서는 이 세 가지 웹 서버 구조의 차이점과 특징, 그리고 다중 처리기 상의 리눅스에서 구현한 방법에 대해서 설명하도록 하겠다.

3.1 RBW의 구현

<그림 2>는 RBW 구조의 기본 동작에 관한 그림이다. RBW 구조[9,10]는 클라이언트에 의해서 행해지는 요구에 대한 병렬성을 감안하여 고안한 구조이다. 우선 프로그램상에서 메인 쓰레드가 되는 Daemon 쓰레드는 지정된 포트를 통해 클라이언트의 TCP 연결 요청을 기다리게 된다. 지정된 포트를 통해 클라이언트의 TCP 연결 요청이 들어오게 되면 메인 쓰레드는 클라이언트의 요구를 처리하는 쓰레드를 생성하게 된다. 생성된 쓰레드는 클라이언트와의 HTTP 연결을 만들게 되고, 이 연결을 통하여 클라이언트의 요구를 받아들이게 된다. 그 후 하나의 요구에 대한 전 과정을 수행하게 되고 이 전 과정이 끝나게 되면 요구를 처리하는 쓰레드는 실행을 종료하게 된다. 이 병행 구조의 웹 서버에서 유의해야 할 사항은 로그 파일의 기록과 캐쉬에 대한 문제이다. 수행되는 여러 개의 쓰레드가 동시에 캐쉬와 로그 파일에 접근할 수 있기 때문에 이 두 자원에 대한 상호배제가 보장되어야 한다.

이 병행 구조의 시스템은 요구되는 파일의 크기가 작거나 서버에 적은 양의 요구가 들어오는 경우에 좋은 성능을 보일 수 있다. 이 때는 동시에 수행되는 쓰레드의 수가 많지 않게 되고, 상호 배제를 위한 오버 헤드도 적게 되기 때문이다. 하지만 동적 프로그램에 대한 요구가 많거나 파일의 크기는 작지만 많은 요구가 한꺼번에 들어오는 경우에는 서버에서 동시에 수행되는 쓰레드의 수가 많게 되므로 예상치 못한 현상이 발생할 수 있다. 기존의 연구[15]에 따르면 원래 UNIX 환경에서는 하나의 프로세스에서 효율적으로 동시에 운영할 수 있는 쓰레드의 수는 한정되어 있고, 만약 그 수 이상의 쓰레드가 하나의 프로세스에서 생성되어 수행이 된다면 그 오버 헤드로 인하여 쓰레드를 유지, 관리하기 위해서 드는 부담은 급격하게 증가하게 된다. 하지만 리눅스 환경에서는 하나의 쓰레드가 하나의 프로세스이기 때문에 그런 문제점이 발생되지는 않는다. 단지 운영 체제 상에서 수행되는 프로세스의 개수가 증가하게 되면 과도한 Context Switching이 일어나게 되고 그에 따른 오버 헤드가 있게 되므로 그로 인해 전체 성능이 저하될 수는 있다.

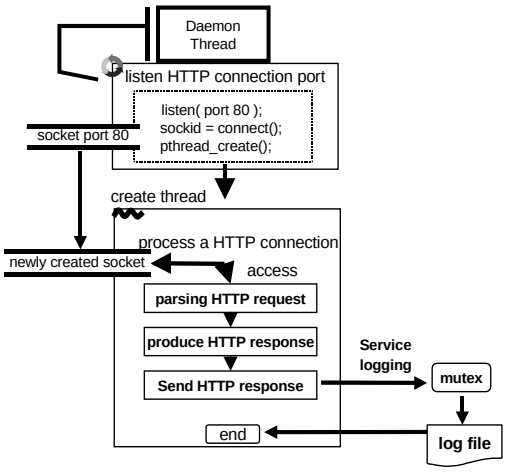


그림 2 RBW 구조의 기본 동작

RBW를 구현하기 위해서 이용한 라이브러리는 Pthread 라이브러리[15]와 Socket 라이브러리[16]이다. Pthread 라이브러리는 쓰레드에 관한 함수를 구현하기 위하여 이용하였고 Socket 라이브러리는 TCP연결을 위해서 이용하였다. 쓰레드는 휘발성 쓰레드로 생성이 된다. 휘발성 쓰레드라는 것은 수행이 종료가 되는 순간 자신이 사용하던 자원들을 시스템에 반납을 하여 다시 생성하는 쓰레드를 위해서 쓸 수 있도록 하는 특성을 가진 쓰레드이다. 이러한 특성을 통해 별다른 처리 없이 자원 낭비를 막을 수 있었다. 공유 자원에 대한 상호 배제를 보장하기 위해서 이용한 것은 mutex이다. 각 쓰레드가 요구 처리 도중에 혹은 요구에 대한 응답을 보낸 후에 캐쉬 혹은 로그 파일에 접근할 때 각 쓰레드는 먼저 mutex를 가져야 한다. 모든 쓰레드 중 mutex를 가진 쓰레드만이 캐쉬 혹은 로그 파일에 접근할 수 있도록 허용함으로 해서 정보들이 유실되지 않도록 하였다.

3.2 TBW의 구현

TBW[14]라는 것은 서버의 여러 수행 단계에 의해서 나올 수 있는 병렬성을 감안하여 고안한 구조이다. 이 구조에서는 서버의 각 단계별로 쓰레드를 할당하여 구성한다. 각 단계에 할당된 쓰레드는 반 영구적으로 수행된다. 즉, 계속 새로운 쓰레드를 생성하는 것이 아니라 웹 서버가 동작하는 동안 하나의 쓰레드가 웹 서버의 수행 단계를 책임지고 수행하게 되는 것이다. 그러므로 이 구조의 웹 서버에서는 고정된 수의 쓰레드만이 수행이 되게 되고 쓰레드 생성, 소멸에 드는 오버헤드는 줄

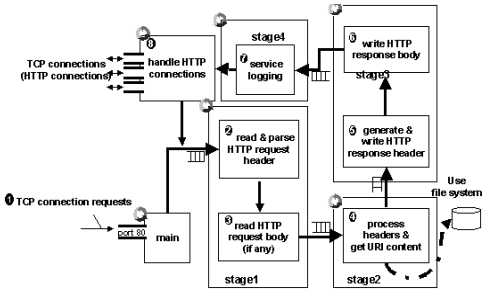


그림 3 TBW의 기본 동작

어들게 된다.

<그림 3>은 TBW의 기본 동작에 대해서 그림으로 나타낸 것이다. 위에서 각 stage에 쓰레드가 할당이 되게 된다. HTTP 프로토콜에서는 웹 서버가 HTTP 요구를 읽고, 해당 웹 문서를 파일 시스템으로부터 읽어오고, HTTP 응답을 연결에 쓰고, 서비스를 기록하는 4 단계의 입출력을 수행한다. 기존의 연구[11]에 의하면 이러한 입출력이 웹 서버 수행 시간의 대부분을 차지한다고 하여 4개의 단계로 나누어 쓰레드를 할당하였다.

단계를 좀 더 세분해서 살펴보면 첫번째 단계는 HTTP 헤더의 분석과 본체의 분석으로 나눌 수 있고 세 번째 단계는 응답의 헤더를 만들어내는 단계와 본체를 만들어내는 단계, 두 가지로 나눌 수 있다. 이외에 main은 TCP 연결 요청을 받는 역할을 하여 그 요청을 넘겨주는 역할을 하고, HTTP 연결 관리 단계는 각각의 HTTP 연결에서 받은 요구에 대한 모든 서비스가 끝난 경우에 그 연결을 해제해주는 기능을 담당한다.

RBW 모델과는 다르게 TBW 모델에서는 캐쉬와 로그 파일의 기록에 이용되는 쓰레드는 항상 하나밖에 없기 때문에 상호 배제에 신경을 쓰지 않아도 된다. 이 모델에서의 병렬화라는 것은 각각의 단계별로 쓰레드를 두는 파이프라인 형식의 병렬화를 말하는 것이므로 하나의 쓰레드가 로그 파일에 기록하는 경우에 다른 쓰레드는 요구를 읽어들이는 기능을 하고 있던지 아니면 응답을 만들어내는 기능을 하는 등의 다른 작업을 하고 있을 것이다. 따라서 RBW 모델과는 다르게 여기서는 캐쉬와 로그 파일은 공유 자원이 되지 못한다. 이 모델에서의 공유 자원이라는 것은 각각의 단계에서 다음 단계로 넘어갈 때 필요하게 된다. 각각의 단계는 다음 단계가 처리해야 할 데이터에 대한 목록을 넘겨 주어야 하는 데 이러한 목적으로 이용되는 것이 큐이다. 각 단계에서는 자신의 책임을 다한 후에 다음 단계에 처리 목록을 넘겨줄 때 그 목록들을 큐에 집어넣게 된다. 그러면 다음 단계의 쓰레드는 그 큐에서 자신이 처리해야 할 목록들을 하나씩 가져와서 자신의 역할을 하게 되는 것이다. 이 과정에서 두 단계의 쓰레드가 동시에 큐에 쓰고 읽게 되면 문제가 발생할 가능성이 있으므로 이런 문제를 해결하기 위하여 상호 배제를 보장하여야 한다.

TBW 모델과 같은 경우에는 각 단계의 시간이 동일하게 걸린다면 큰 문제 없이 수행되게 될 것이다. 하지만 만약 각 단계에서 걸리는 시간이 서로 다르게 된다면 문제가 발생하게 된다. 클라이언트의 요구가 많은 서버를 생각해 볼 때 각 단계에서 걸리는 시간이 서로 다르면 그 시간의 차이가 누적되게 되어 나중으로 갈수록 처리 시간에 점점 크게 영향을 끼치게 된다. 또한 이런 모델에서는 각각의 요구가 다른 요구에 영향을 끼치는 정도가 RBW 모델에서보다는 크게 나타난다. 예를 들어 저장 매체가 그리 좋은 성능이 아니라고 가정할 때 파일을 읽는 단계에서 아주 큰 파일을 읽게 되면 그 전 큐에서 대기하고 있는 작업들은 그 파일이 모두 읽혀질 때까지 기다려야만 한다. 하지만 RBW 모델에서는 이러한 경우에도 서로 독립적으로 쓰레드들이 수행될 수 있기 때문에 다른 쓰레드는 그동안 남아있는 작업을 수행할 수 있으므로 TBW 모델보다는 큰 영향을 받지 않게 된다.

큐에서의 대기 시간이 누적되는 문제점을 개선하기 위하여 TBW 모델을 조금 변형하는 것이 가능하다. 대기 시간이 누적되는 문제점은 각 단계 처리 시간의 불균형 때문에 일어나는 것이기 때문에 각각의 단계를 수행하는 쓰레드의 개수를 조절하여 이러한 불균형을 조금 개선할 수 있을 것이다. 쓰레드의 개수를 조절함으로 해서 하나의 쓰레드가 작업하는 동안 다른 쓰레드들은 큐에서 나머지 요구들에 대한 작업을 계속할 수 있도록 하였다.

<그림 4>는 개선된 모델의 모습을 나타내는 것이다. 각 단계마다 하나의 쓰레드를 두는 경우에는 그 쓰레드가 작업하는 동안 큐에서는 많은 요구들이 기다리고 있어야 했었는데,

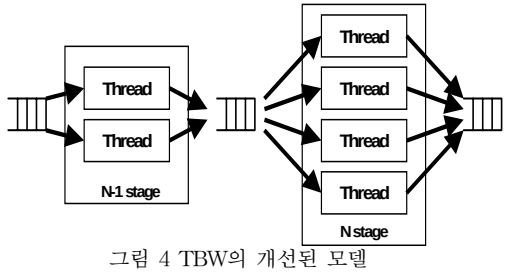


그림 4 TBW의 개선된 모델

각 단계마다 쓰레드의 개수를 조절함으로 해서 하나의 쓰레드가 작업하는 동안 다른 쓰레드들은 큐에서 나머지 요구들에 대한 작업을 계속할 수 있도록 하였다.

이런 방법에 의하면 큐를 동시에 접근할 수 있는 쓰레드의 개수가 늘어나 상호 배제를 보장하는 오버 헤드는 늘어나지만 큐에서의 대기 시간이 누적되는 문제점은 조금 개선이 되게 된다. 앞에서 TBW 구조에서는 캐쉬와 로그 파일이 공유 자원이 아니라는 얘기를 하였다. 하지만 TBW 구조를 개선한 모델에서는 각 단계마다 동시에 수행되어지는 쓰레드의 개수가 하나가 아니기 때문에 조금 달라진다. 예를 들어서 로그 파일을 쓰는 단계에서 2개 이상의 쓰레드를 둔다고 하면 같은 로그 파일에 동시에 두 개의 쓰레드가 쓰는 작업을 할 수 있으므로 경쟁 상태가 일어날 수 있다. 그러므로 이러한 경우에는 RBW 구조에서와 마찬가지로 mutex를 이용하여 상호 배제를 보장해주어야 한다. 쓰레드 개수에 있어서 본 논문에서는 CPU 4개짜리 컴퓨터를 이용하고, CPU 부하가 많이 걸리는 CGI 요구를 중심으로 실험을 하기 때문에 그에 맞게 쓰레드의 개수를 결정하였다. 그렇지만, 다른 환경에서는 실험에 의해서 경험적으로 쓰레드의 수를 맞추어 보다 알맞게 운용할 수 있을 것이다.

TBW를 구현하기 위해서 이용한 라이브러리는 RBW와 마찬가지로 Pthread 라이브러리[15]와 Socket 라이브러리[16]이다. 쓰레드의 종류와 상호 배제를 위한 도구 또한 각각 휘발성 쓰레드와 mutex로 RBW와 마찬가지이다. 기본적인 TBW 구조에서는 각각의 단계마다 하나의 쓰레드가 할당되지만 보다 성능을 향상시킨 TBW 구조에서는 각각의 단계마다 할당되는 쓰레드의 개수를 조절해야 한다. 본 논문의 실험을 위한 구조에서는 CPU 개수에 맞추어 4개의 쓰레드를 CGI를 수행하는 데 할당하였다. 하지만 이것은 환경에 따라 달라질 수 있을 것이다. 큐에서의 상호 배제를 보장하기 위해서 mutex를 이용하였다. 큐에 넣을 때나 혹은 큐에서 요구를 가지고 올 때 mutex를 가진 쓰레드만 접근을 할 수 있도록 하여, 큐에서의 정보 유실을 막을 수 있다. 큐는 Linked List로 구현이 되었고, List의 Head에서 요구를 가지고 올 수 있고, Tail에서 요구를 입력할 수 있도록 되어 있다. 만약 List에 남아 있는 요구가 없는 데 요구를 가지고 와야 할 때에는 요구가 입력 될 때까지 기다리도록 하였다.

3.3 TPW의 구현

<그림 5>는 TPW[11,13]의 기본 동작을 설명하는 그림이다. 그림을 보면 알 수 있지만 기본 모델은 TBW 모델과 거의 비슷하다.

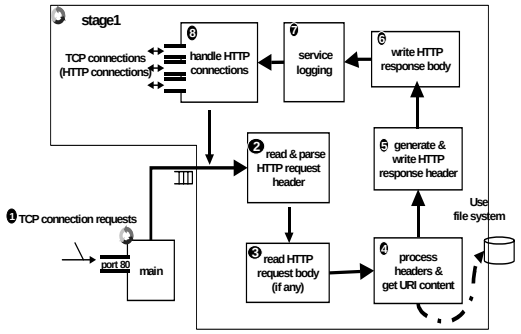


그림 5 TPW의 작동 원리

하지만 TBW 모델에서는 여러 단계가 있고 그 단계들이 파이프라인 형식으로 연결된 데 비해 TPW에서는 한 요구가 서버에 도착하면 그 요구에 대해서 서버가 서비스 해주는 모든 일이 하나의 단계로 되어 있다. 즉, 여러 단계가 하나의 단계로 합쳐졌으며, 그 단계 사이의 큐가 없어진 것이다. RBW와는 하나의 쓰레드가 하나의 요구를 처리하는 데 할당된다는 점에서는 비슷하지만 RBW에서는 하나의 쓰레드가 하나의 요구를 처리한 후에 그 쓰레드의 수행을 종료한 데 반해 TPW에서는 메인 쓰레드가 받는 요구들을 큐에 넣어서 순서화 시킨 후에 몇 개의 정해진 쓰레드 개수만큼 수행되면서 큐에서 요구들을 가지고 오는 점이 다르다. 즉, 이 모델에서의 쓰레드라는 것은 반영구적이고, 고정된 수의 쓰레드 개수만이 수행이 되는 것이 RBW와는 다른 점이라고 할 수 있다.

RBW에서는 여러 쓰레드가 캐쉬와 로그 파일에 접근할 수 있었기 때문에 캐쉬와 로그 파일이 공유 자원이 됐었다. 그에 반해 TBW에서는 캐쉬와 로그 파일은 어느 순간 하나의 쓰레드만이 접근할 수 있어서 공유 자원이 되지 않았고, 각 단계에서 그 다음 단계로 작업들을 넘겨주는 데 이용되는 큐가 공유 자원이 됐었다. TPW에서는 TBW와는 달리 이용되어지는 큐의 수가 단 한 개이다. 즉, 메인 쓰레드가 받는 클라이언트의 요구를 서버의 서비스 단계가 처리할 수 있도록 넘겨주는 데 이용하는 큐 단 한 개만이 존재한다. 그러므로 이 단 한 개의 큐에서만 상호 배제를 보장해주면 된다. 또한 TPW에서는 처음에 정해진 개수의 쓰레드만이 수행이 된다. 이 정해진 개수의 쓰레드가 하나 이상이라면 그 하나 이상의 쓰레드들은 동시에 캐쉬와 로그 파일을 접근할 수 있기 때문에 이 두 자원에 대해서도 상호 배제를 보장해주어야 한다.

TPW를 구현하기 위해서 이용한 라이브러리는 앞의

두 구조의 웹 서버와 마찬가지로 Pthread 라이브러리[15]와 Socket 라이브러리[16]이다. 쓰레드의 종류와 상호 배제를 위한 도구 또한 각각 휘발성 쓰레드와 mutex로 앞의 두 구조의 웹 서버와 마찬가지이다. 본 논문에서는 요구를 처리하기 위한 쓰레드의 수를 10개로 할당하였다. 본 논문의 환경에서는 5개에서 16개 정도 사이가 가장 좋은 성능을 보임을 실험을 통하여 알 수 있었기 때문에 10개의 쓰레드 개수로 정하였지만 각 환경에 따라서 다르게 생성할 수 있을 것이다. 이 외 동시에 연결할 수 있는 TCP 연결의 수, 큐는 TBW와 동일한 방법으로 구현이 되었다.

4. 실험 및 분석

이 장에서는 위에서 제시한 세 가지 웹 서버 구조를 직접 구현하여 수행한 실험 결과와 그 결과에 대한 분석을 보이도록 하겠다. 본 논문에서 성능의 척도로 사용한 것은 웹 서버의 Throughput이다. 웹 서버의 Throughput이라는 것은 단위 시간당 얼마만큼의 요구를 처리할 수 있는가를 나타내는 기능을 한다. 즉, 단위 시간당 얼마만큼의 클라이언트를 지원할 수 있는가를 나타내는 것인데 웹 서버에서는 자신이 처리할 수 있는 부하량 이상이 오면 문서의 신뢰성이나, 클라이언트가 문서를 받을 때까지의 시간에 악영향을 끼치게 되므로 웹 서버의 Throughput은 성능 척도의 기준이 될 수 있다. 기존의 연구들이 적은 개수의 파라미터를 이용하였다는 문제점을 갖고 있어서 본 논문에서는 여러 파라미터를 조정하면서 실험을 하였는데, 이용한 파라미터들은 CPU 개수, 웹 서버 구조, 동적 프로그램(CGI) 요구의 비율, 파일 크기, 부하량 등이다. 4.1절에서는 실험 환경에 대해 설명하도록 하겠고, 4.2절에서는 실험 결과를 통해 살펴본 각 웹 서버 구조의 특징을 살펴보도록 하겠다. 4.3절에서는 정적 파일에 대한 요구만 주어졌을 때의 실험 결과를 보이도록 하겠고 4.4절에서는 동적 프로그램에 대한 요구만 주어졌을 때의 실험 결과를 보이도록 하겠다. 4.5절에서는 정적 파일과 동적 프로그램에 대한 요구 전반에 걸친 실험 결과를 보이도록 하겠고 마지막으로 4.6절에서는 전체 실험 결과를 정리하도록 하겠다.

4.1 실험 환경

<그림 6>은 실험 구조를 설명하는 그림이다. 실험에 이용된 구조를 살펴보면 크게 세 가지로 분류가 된다. 첫째는 요구를 보내주는 부하 생성기, 그리고 두 번째로는 그 요구들을 처리하는 웹 서버이고 마지막으로 세 번째는 네트워 부분이다. 부하 생성기가 탑재된 컴퓨터는 운영 체제로 리눅스(커널 버전 2.2.12)를 사용하고 있고, 32MB의 메모리를 사용하고 있다. CPU는 한 개를 가지고 있으며 기종은 펜티엄 133Mhz이다. 다음으로 웹 서버가 탑재된 컴퓨터는 운영 체제로 리눅스(커널 버전 2.2.12)를 사용하고 있고, 256MB의 메모리를 사용하고 있다. SMP(Symmetric Multiprocessing) 구조를 가진 4개의 CPU를 가지고 있으며 기종은 각각 펜티엄 II Xeon 450Mhz이다. 네트웍은 10 Base-T LAN 환경에서 실험을 하였다. 각각의 요구를 생성하거나 혹은 그 요구를 처리하기 위해서 쓰레드가 생성되는데, 쓰레드를 생성하기 위해서 Pthread 라이브러리[16]를 이용하였다.

웹 서버가 탑재된 컴퓨터의 운영 체제로 리눅스를 이용한 이유는 앞에서도 잠깐 언급한 바와 같이 리눅스의 쓰레드 특성을 이용하기 위해서 이다. 리눅스에서는 보통 프로세스가 fork( )를 이용하여 생성되는 반면 쓰레드는 clone( )을 통하여 생성된다[17]. 두 System Call은 거의 비슷한 기능을 하지만 다른 점이 있다면 clone( )을 통해 생성된 프로세스는 공유 자원을 가질 수 있는 반면 fork( )를 통해 생성된 프로세스는 그럴 수 없다는 것이 다른 점이다. 이런 점만 제외한다면 결국 프로세스와 쓰레드는 거의 같은 개념으로 볼 수 있다. 이런 리눅스의 쓰레드 특성을 통해 리눅스에서는 쓰레드를 각 CPU로 할당하는 스케줄링을 운영 체제상에서 제공을 하여 생성된 쓰레드들을 병렬로 수행이 될 수 있게 한다. 그러므로 여러 개의 요구 처리가 동시에 수행이 될 수 있다.

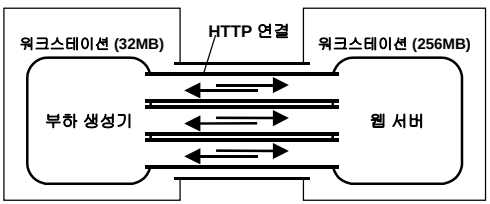


그림 6 실험 구조

4.2 실험을 통한 각 병행 웹 서버의 특징 분석

<그림 7>은 RBW 구조에 대해 1개의 CPU를 사용하는 환경하에서 1초당 35개의 요구를 서버 측으로 보내고, 그 중 10%를 CGI 에 대한 요구로 보냈을 때 서버측에서 걸리는 시간에 대한 그래프이다. 정적 파일의 크기는 115K를 이용했다. 위 그래프는 RBW의 문제점을 보여주고 있다. 실제로 시간 분석을 해보면 CGI에 대한 요구가 적은 비율이기 때문에 CPU 에 대한 로드

는 적게 되지만 파일의 크기가 비교적 큰 데이터에 대한 요구이기 때문에 네트웍으로 보내는 데 걸리는 시간이 가장 크게 된다. 결국 서버에서 클라이언트로 가는 네트웍 라인은 하나이므로 병목 현상이 일어나게 된다. 이런 병목 현상으로 인해 동시에 수행되는 프로세스의 개수가 늘어나고 스케쥴링에 의하여 먼저 받은 요구에 대한 응답이 늦어지게 되는 경우가 생기게 된다. 그래프에서 원으로 둘러 싸인 점들을 보면 서버에서 서비스를 시작하는 시간이 같은 다른 점들에 비해 서비스를 종료하는 시간은 늦음을 볼 수 있다. 즉, 이 점에 해당하는 클라이언트는 그만큼 다른 클라이언트보다 서비스 받는 시간이 늦게 된다. 이렇듯 클라이언트간의 latency가 다양한 것은 웹 서버에 있어서 큰 문제점으로 대두 될 수 있게 된다.

<그림 8>은 TBW의 수행 시간에 대한 그래프로 실험 조건은 위 RBW에 적용되었던 것과 같다. TBW의 구조에서는 각 큐에서 클라이언트의 요구에 대해 순서를 맞추어주기 때문에 RBW에서 나타나는 문제점인 몇몇 클라이언트에게 가해지는 서비스의 불균형은 거의 나타나지 않는다. 즉, 비슷한 시간에 서버에 주어진 클라이언트의 요구에 대해서 어떤 요구는 먼저 수행되고 어떤 요구는 아주 늦게 수행이 되는 그런 불균형은 나타나지 않고 비슷하게 끝나던지 아니면 서비스 시간의 차이가 나도 RBW에서 나타나는 것만큼 크게는 나타나지 않게 된다. <그림 8>을 보면 이러한 현상이 증명된다. <그림 8>은 서버측에서 걸리는 시간에 대한 그래프로 실험 조건은 위 RBW에 적용되었던 것과 같다. x축이 시작 시간이고 y축은 서비스 종료 시간이다. 비슷한 x값을 가진 점들 중에서 y 값이 갑작스럽게 커지는 값은 보이질 않는다. 즉, RBW의 문제점인 Latency의 다양성은 나타나지 않는다. 하지만 이러한 구조에서의 문제점은 시간이 오래 걸리는 단계에서의 쓰레드 개수를 조절해도 성능이 좋아지는 데 한계가 있다는 것이다. 네트웍과 같은 입출력을 고려할 때 이와 같은 부분은 쓰레드의 개수를 늘려도 크게 좋아지지는 않게 된다. 또한 큐가 이용이 되므로 큐 앞에서 수행이 되는 쓰레드의 수행 시간이 큐에 있는 다른 요구의 수행 시간에 영향을 끼치게 된다는 문제점이 있다. 이런 문제점으로 인해서 <그림 8> 그래프 중간에 끊기는 현상이 나타나는 것이다. 즉, 전에 수행이 된 요구의 수행 시간이 누적되면서 다른 요구들은 그만큼 종료 시간이 늦어지는 것이다.

<그림 9>는 TPW의 수행 시간에 대한 그래프로 실험 조건은 위 RBW에 적용되었던 것과 같다. 우선 TPW는 어떤 시간에 수행되고 있는 쓰레드의 개수가 정해져 있으므로 RBW에서 나타나는 문제점인 Latency의 다양성은 크게 찾아볼 수 없다. 비슷한 시간에 요구를 보낸 클라이언트들은 거의 비슷한 시간에 끝남을 볼 수 있다. 또한 TBW에서 보이는 문제점인 큐에서의 대기 시간 누적 또한 찾아보기 힘들다.

TPW 구조에서는 큐를 하나만 이용하기 때문에 그리고 그 큐마저도 웹 서버의 단계 사이에서 존재하는 큐

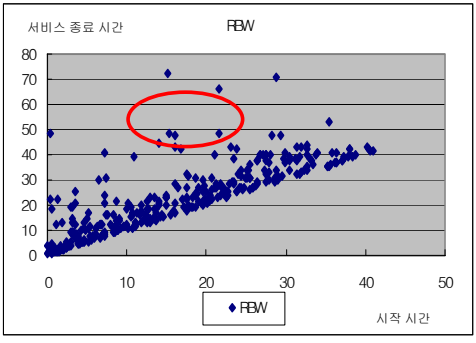


그림 7 RBW의 수행 시간에 대한 그래프

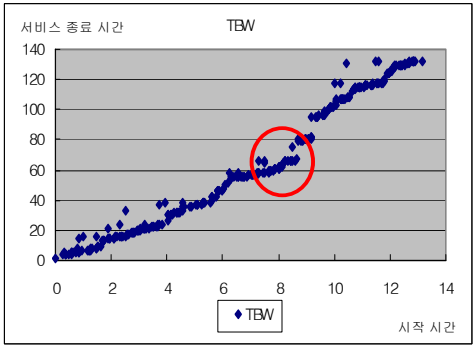


그림 8 TBW의 수행 시간에 대한 그래프

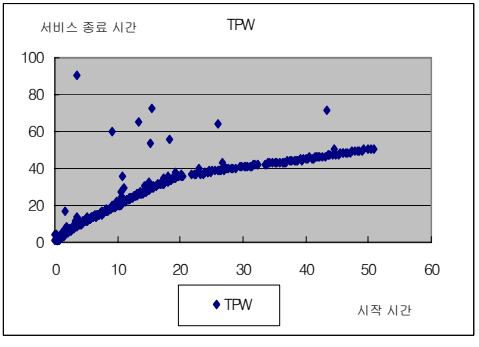


그림 9 TPW의 수행 시간에 대한 그래프

가 아니라 단순히 클라이언트의 요구를 저장하기 위한 큐이므로 큐에서의 대기 시간 누적은 있을 수 없게 되는 것이다. 이 모델에서의 문제점으로 볼 수 있는 것은 Latency의 다양성과 서비스 시작 시간 자체가 늦어지게 되는 경우이다. 물론, RBW와 비교한다면 좋아졌지만, 여전히 조금은 Latency의 다양성이 남아있고 이것은 어떤 클라이언트에게는 문제점으로 작용할 수 있다. 그리고 서비스 시작 시간이 늦어진다는 것은 위 그림에서 크게 나타나지는 않지만 다른 환경에서 실험을 했을 때 나타난 현상이다. 정해진 모든 쓰레드의 개수만큼 수행이 되고 있는 동안 클라이언트로부터 요구가 들어오면 그 요구를 서버가 서비스해주기 위해서는 기존에 수행되는 쓰레드가 끝나기를 기다려야 할 것이다. 그러므로 그만큼 큐에서 작업은 대기해야 하기 때문에 시작 시간이 늦어지는 것이다.

비록 위에서 실험 결과로 제시된 결과물이 하나의 조건에 한정되어 있지만 다른 조건에서도 비슷한 결과가 나왔다. 조금이라도 웹 서버에 과부하가 걸리게 되면 RBW에서는 수행이 되는 쓰레드의 개수가 증가하게 되므로 Latency의 다양성이 나타나게 되고, TBW에서는 큐에서 기다리는 요구가 누적이 되므로 대기 시간의 누적 현상이 나타나게 된다. 그리고 TPW 또한 요구를 받아들이는 큐에서 기다리는 요구의 수가 증가하게 되므로 시작 시간이 늦어지는 문제점이 보이는 것이다.

4.3 정적 파일의 요구에 대한 실험 결과

이 실험에서 이용된 파일들의 크기는 각각 500바이트, 50K바이트, 115K 바이트이다. 기존의 연구 결과[8]를 살펴보면 웹 서버에 전달이 되는 요구들의 90%를 훨씬 넘는 비율이 100바이트에서 100K바이트 까지라고 한다. 이런 이유로 인해 이 범위 안에서 작은 파일과 큰 파일 그리고 중간 크기의 파일을 WebStone이라는 프로그램에서 이용되는 파일 중에서 사용하였다.

<그림 10>은 115K 바이트 크기의 정적 파일 요구에 대한 TPW의 성능을 CPU 개수에 따라 보여주는 그래프이다. 위 그래프에 의해서 보여주는 것은 정적 파일의 요구에 대한 웹 서버의 성능은 CPU 개수에 따라 크게 차이가 나지 않는다는 것이다.

RBW, TBW에서도 같은 결과과 나옴을 실험을 통하여 확인할 수 있었다. 정적 파일의 요구를 처리하는 데 있어서 가장 많이 시간이 걸리는 부분은 네트웍 부분이다. 이런 네트웍 병목 현상은 CPU의 개수가 많아진다 해도 큰 이득을 볼 수 없다

정적 파일의 요구를 처리하는 데 있어 CPU를 사용하는 부분은 요구를 분석하는 부분을 제외하고는 거의 없는데 요구를 분석하는 부분 또한 CPU에 큰 무리가 가는 부분은 아니기 때문에 한 개의 CPU로도 충분히 처리할 수 있게 된다. 그렇기 때문에 CPU의 개수를 늘린다고 하더라도 큰 이득을 볼 수는 없는 것이다.

그림 10 115K 바이트 크기 정적 파일의 요구에 대한 TPW의 성능

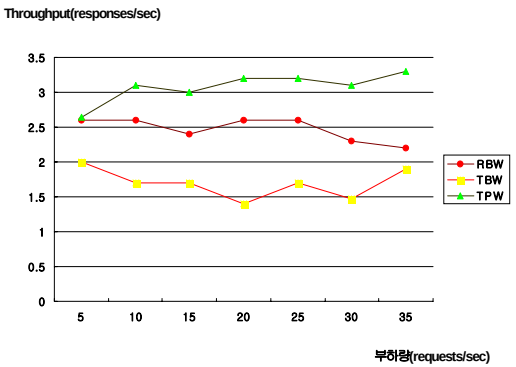


그림 11 115K 바이트 크기의 정적 파일 요구에 대한 웹 서버의 성능 비교

<그림 11>은 115K 바이트 크기의 정적 파일 요구에 대해서 CPU의 개수가 4개일 때 각 웹 서버의 성능을 비교한 그래프이다. 전체적으로 TPW가 가장 좋은 성능을 보임을 알 수 있다. 정적 파일의 요구에 대해 웹 서버의 성능을 좌우하는 것은 네트웍이다. 조금이라도 네트웍 병목 현상을 줄일 수 있으면 그 웹 서버의 성능이 좋아지게 된다. TBW는 네트웍 병목 현상 뿐만 아니라 네트웍 병목 현상이 큐의 병목 현상까지 일으켜 다른 웹 서버에 비해 좋지 않은 성능을 보이게 된다. RBW는 큐의 병목 현상은 없지만 다른 웹 서버에 비해 한 번에

네트웍으로 보내려는 쓰레드의 개수가 많기 때문에 그로 인한 병목 현상으로 TPW보다는 나쁜 성능을 보이게 된다. TPW는 비록 네트웍 병목 현상이 있기는 하지만 동시에 수행이 되는 쓰레드의 개수를 조절해주기 때문에 조금은 줄일 수 있다. 그로 인하여 다른 웹 서버에 비해 좋은 성능을 보이고 있다. 위 그림은 115K 바이트 크기의 정적 파일 요구에 대해서만 비교를 한 그래프인데, 파일의 크기가 작아질수록 병목 현상이 줄어들어 점점 성능의 차이는 줄어들고, 500바이트 크기의 정적 파일에 대해서는 오히려 TBW가 RBW보다 약간은 좋은 성능을 보임을 알 수 있었다. 하지만 그 성능 차는 그렇게 크게 나타나지 않아 큰 의미가 있지는 않다.

위의 실험 결과들을 볼 때 정적 파일 요구에 대해서 웹 서버가 처리하는 경우에는 CPU를 많이 사용하지 않기 때문에 CPU 개수가 많아지더라도 성능이 향상되지 않음을 볼 수 있다. 즉, 이 경우에는 다중 처리기를 사용하더라도 큰 이득을 볼 수는 없게 된다. 그리고 세 개의 웹 서버를 비교하였을 때 TPW 가 가장 좋은 성능을 보임을 알 수 있었다. 이는 RBW가 네트웍에서 큰 병목 현상을 보이고, TBW는 네트웍 병목 현상이 큐의 병목 현상까지 발생시켜 성능이 떨어지는 반면 TPW는 쓰레드의 개수가 조절되어서 네트웍 병목 현상을 조절해주고, 큐가 없기 때문에 큐의 병목 현상 또한 나타나지 않게 된다. 이런 이유로 인해 TPW가 가장 좋은 성능을 나타내는 것이다.

4.4 동적 프로그램의 요구에 대한 실험 결과

동적 프로그램의 요구에 대한 실험으로 이용된 CGI 프로그램은 25clock 의 수행 시간을 가진 프로그램이다. 실험 환경에서 웹 서버가 탑재된 컴퓨터는 100 clock 이 1초를 나타내기 때문에 결국 이 프로그램의 수행 시간은 0.25초 정도의 프로그램이다.

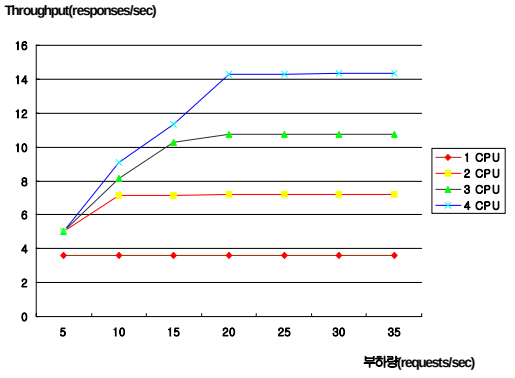


그림 12 동적 프로그램의 요구에 대한 TPW의 성능

<그림 12>는 동적 프로그램의 요구가 서버로 전해졌을 때 서버의 성능은 어떻게 나타나는 지 CPU 개수에 따라 측정한 그래프이다. 동적 프로그램은 물론 그 프로그램 자체에서 CPU를 사용하는 비율이 정적 파일의 요구에 비해 크기도 하겠지만 새로운 프로세스를 만들어내는 데 CPU를 사용하는 비율이 커서 CPU 개수에 따라 큰 차이를 보이게 된다. 정적 파일의 요구에 대해서는 비록 CPU의 개수가 4개라고 할 지라도 서버가 CPU를 사용하는 비율이 높지 않기 때문에 4개 모두가 병렬로 수행이 되는 경우는 거의 없다. 하지만 동적 프로그램의 요구에 대해서는 서버가 CPU를 사용하는 비율이 정적 파일의 요구에 비해서는 현격하게 높기 때문에 4개의 CPU 모두가 병렬로 수행이 되는 경우가 많게 된다. 그렇기 때문에 위의 그래프와 같이 CPU의 개수에 따라 큰 폭으로 서버의 성능이 증가됨을 볼 수 있다. 이와 같은 현상은 다른 웹 서버의 구조에서도 똑같이 나타난다.

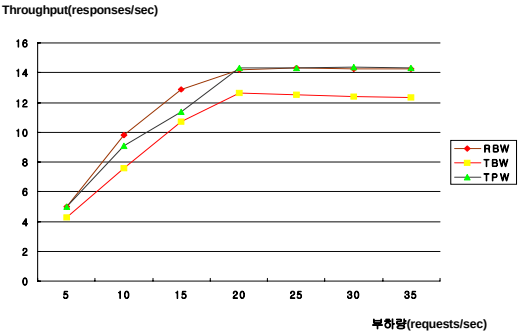


그림 13 동적 프로그램의 요구에 대한 웹 서버의 성능 비교

<그림 13>은 동적 프로그램의 요구에 대한 웹 서버의 성능을 비교한 그래프이다. 그래프를 보면 알 수 있지만 전체적으로는 비슷한 성능을 보임을 알 수 있다. 위 그래프는 4개의 CPU를 사용한 경우인데, 각 구조마다 실질적으로 4개의 CPU 모두에게 쓰레드가 할당이 될 수 있는 구조이기 때문에 병렬로 수행이 가능하다. 그렇기 때문에 4개의 CPU 모두를 사용한 성능이 나타나는 것이다. 큰 차이가 아니기는 하지만 다른 구조의 웹 서버에 비해 TBW 구조의 성능이 안 좋은 것은 CGI를 수행하는 단계의 전 큐에서 병목 현상이 발생하기 때문이다.

위의 결과들을 통해 알 수 있는 것은 우선 동적 프로

그램의 요구를 웹 서버가 처리하는 데 있어서 CPU의 개수가 많을 때의 성능이 CPU의 개수가 적을 때의 성능보다 훨씬 좋은 성능을 보인다는 것이다. 정적 파일 요구를 처리하는 데 있어서는 CPU를 사용하는 비율이 높지 않으므로 성능이 좋아지는 것을 볼 수 없었지만 동적 프로그램을 처리하는 데 있어서는 새로운 프로세스를 생성할 때 CPU를 많이 사용하게 되므로 CPU의 개수가 많은 경우가 좋은 성능을 나타내는 것이다. 그리고 웹 서버 구조 사이에서는 CPU 4개를 모두 사용할 때 TBW의 성능이 가장 안 좋게 나타나는 것을 볼 수 있었는데 이는 CGI를 수행하는 단계의 전 큐에서 병목 현상이 일어나기 때문이다.

4.5 혼합한 형태의 요구에 대한 실험 결과

이 절에서는 동적 프로그램에 대한 요구와 정적 파일의 요구를 혼합하여 요구를 보낼 때 웹 서버의 성능은 어떻게 측정이 되는 지를 알아보도록 하겠다. 기본적으로 동적 프로그램의 요구에서 사용하는 CGI 프로그램과 정적 파일의 요구에서 이용되는 파일은 위에서 이용한 프로그램과 파일을 이용하도록 했다. CGI 요구의 비율과 파일의 크기, CPU의 개수 등이 파라미터로 이용이 된다.

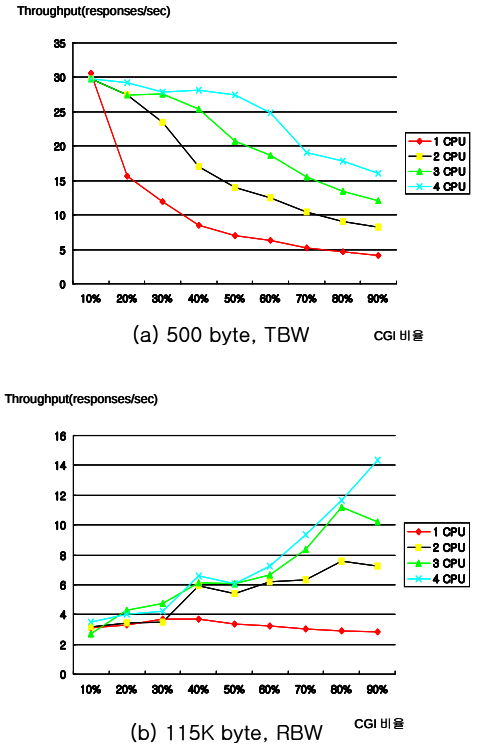


그림 14 혼합한 요구에 따른 웹 서버 성능(RBW)

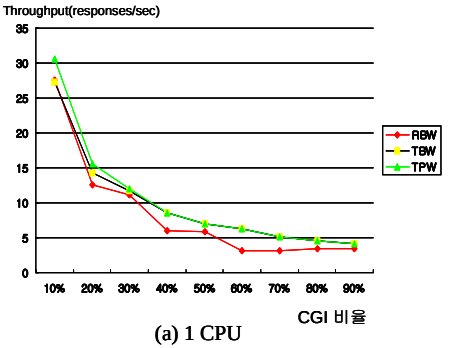
<그림 14>의 그래프는 정적 파일의 요구와 동적 프로그램의 요구를 혼합한 형태의 요구에 웹 서버의 성능이 성능을 나타내는 실험 결과를 나타내는 그래프이다. (b)는 파일 크기가 115K 바이트 이고, 웹 서버로 RBW를 이용하였을 때의 그래프이다. (a)의 그래프는 파일 크기가 500 바이트 이고, 웹 서버로 TPW를 이용하였을 때의 그래프이다. (a)의 그래프는 이용된 파일의 크기가 작기 때문에 CPU 개수에 따른 성능의 차이를 쉽게 볼 수 있다. 하지만 (b)에서 이용된 파일은 115K 바이트의 큰 파일이므로 CGI를 요구하는 비율이 작을 때는 즉, 큰 파일을 요구하는 비율이 클 때는 CPU 개수에 따른 성능의 변화는 크게 보이지 않는다. 오히려 네트웍 병목 현상으로 거의 비슷한 성능을 보임을 알 수 있다.

이와 같은 현상은 CGI 에 대한 요구가 30%가 될 때까지 지속된다. CGI에 대한 요구가 그 이상이 될 때가 되어서야 CPU 개수에 따른 성능의 변화를 볼 수 있고, CGI에 대한 요구가 더할수록 성능의 차이는 점점 커짐을 알 수 있다. 또한 (a)는 CGI의 비율에 커짐에 따라 성능이 저하되지만 (b)는 성능이 향상되는 것을 볼 수 있다. 이것은 115K 바이트 크기의 파일을 웹 서버가 처리하는 시간이 CGI를 처리하는 시간보다 긴 시간을 요구함을 알 수 있다. 그로 인해 (b)는 CGI의 비율이 커질수록 Throughput은 더 좋게 나오는 것이다.

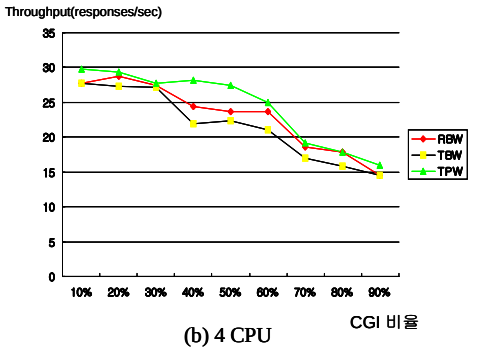
<그림 15>은 CGI 요구와 500 바이트 크기의 정적 파일 요구를 혼합한 요구를 이용할 때의 성능을 비교한 그래프이다. (a)는 한 개의 CPU를 이용할 때의 성능 비교 그래프이다. 이 그래프를 보면 이런 환경에서는 동시에 수행이 되는 쓰레드의 개수가 가장 많은 RBW의 성능이 가장 안 좋음을 알 수 있다. 그에 반해 TPW는 고정된 수의 쓰레드만이 동작을 하기 때문에 웹 서버에 과부하가 걸리는 것을 막아준다. 그로 인해 다른 웹 서버보다는 좋은 성능을 보인다. TBW는 그래프에서 자세히 나타나지는 않지만 TPW보다 조금 좋지 않은 성능을 보인다. 그 이유는 요구 처리 과정 중 CPU를 가장 많이 이용하는 부분인 요구 분석하는 단계에서 병목 현상이 발생하고 그 병목 현상으로 인해 그 전 큐의 작업들에게까지 영향을 미치기 때문이다.

이와 같은 큐에서의 병목 현상으로 인하여 TPW보다는 조금 떨어진 성능을 보이게 되었다.

(b)는 네 개의 CPU를 이용할 때의 성능 비교 그래프이다. 1개의 CPU를 이용하였을 때의 그래프와는 약간 다름을 알 수 있다. RBW는 비록 동시에 수행이 되는 쓰레드의 개수는 많지만 그 쓰레드가 4개의 CPU로 분산이 되기 때문에 TBW 구조보다 오히려 좋은 성능을



(a) 1 CPU



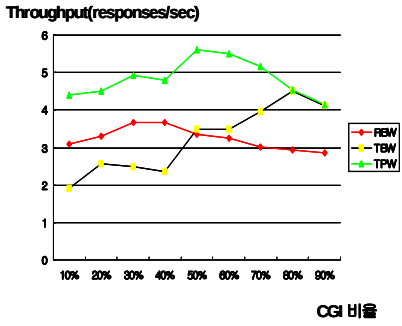
(b) 4 CPU

그림 15 CGI 요구와 500 바이트 파일 요구를 혼합한 요구에 대한 웹 서버 성능 비교

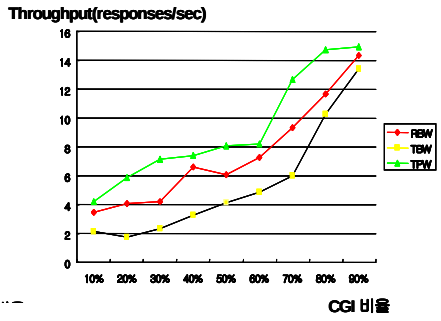
보임을 알 수 있다. TBW 구조의 웹 서버는 요구 분석을 하는 단계에 한 개의 쓰레드만 할당이 되기 때문에 병목 현상이 크게 줄어들지는 않는다. 그에 따라 그 전 큐에서의 병목 현상 또한 그대로 남아있는 것이다. 이로 인해 다른 웹 서버 구조보다 안 좋은 성능을 보이고 있다. TPW는 여전히 수행되는 쓰레드 수의 조절과 쓰레드 생성, 소멸의 오버헤드를 줄임으로 해서 좋은 성능을 보인다.

<그림 16>은 CGI 요구와 115K 바이트 크기의 정적 파일 요구를 혼합하여 요구를 보냈을 때의 성능을 비교한 그래프이다. 그 중 (a)는 한 개의 CPU를 이용할 때의 그래프이다. CGI의 비율이 작을 때는 TPW, RBW, TBW 순으로 성능이 좋음을 알 수 있고, CGI의 비율이 클 때는 TPW, TBW, RBW 순으로 성능이 좋음을 알 수 있다. CGI의 비율이 작을 때를 고려하면 RBW 구조의 병목 현상은 네트웍에서만 나타난다. 하지만 TBW 구조에서는 네트웍 병목 현상이 큐의 병목 현상 까지 일으키게 된다. 이로 인해 TBW 구조의 성능이 RBW 구조의 성능보다 나쁘게 된다. TPW는 네트웍 병목 현상이 있기는 하지만 수행되는 쓰레드 수를 한정지음으로 해서 조절을 할 수 있고 큐에서의 병목 현상 또한 존재하지 않는다. 그로 인해 다른 웹 서버보다 좋은 성능을 보인다. CGI의 비율이 클 때를 고려하면 RBW 구조의 웹 서버는 높아진 CGI의 비율과 많은 수의 쓰레드에 의해 CPU에 과부하를 주게 된다. 그로 인해 가장 안 좋은 성능을 보이게 된다. TBW 구조는 줄어든 네트웍 병목 현상으로 인해 큐의 병목 현상 또한 줄어들게 된다. 물론 CPU에 과부하가 걸리기는 하지만 고정된 수의 쓰레드로 인해 과부하를 조절하게 된다. 단지 CGI를 처리하는 단계에서 오래 걸리기 때문에 그 전 큐에서의 병목 현상이 존재하기는 한다. 그로 인해 TPW보다는 나쁜 성능을 보인다. TPW는 여전히 CPU의 부하를 조절하는 역할을 하는 고정된 쓰레드와 큐에서의 병목 현상을 없앤 이유로 가장 좋은 성능을 보인다.

<그림 16>의 (b)는 4 개의 CPU를 이용할 때의 그래프이다. TPW가 가장 좋음을 알 수 있고, 다음으로는 RBW 그리고 TBW가 가장 안 좋음을 알 수 있다. 그



(a) 1 CPU



(b) 4 CPU

그림 16 CGI 요구와 115K 바이트 파일 요구를 혼합한 요구에 대한 웹 서버 성능 비교

래프의 모양이 <그림 15>의 (b)와 비슷한데, 그 이유 또한 비슷하다. 즉, RBW는 쓰레드를 4개의 CPU로 나눌 수 있기 때문에 조금이라도 성능은 좋아지지만 큐에서의 병목 현상이라는 것은 CPU의 개수에 큰 영향을 받는 것이 아니므로 다른 웹 서버보다 떨어지는 성능을 보이게 되는 것이다.

위의 실험 결과를 통해 정적 파일 요구와 동적 프로그램 요구를 혼합한 요구를 처리하는 경우에도 전체적으로 CPU의 개수가 많아질수록 성능이 향상됨을 볼 수 있었다. 이는 동적 프로그램 요구에 대한 실험 결과와 마찬가지 이유에서이다. 즉, 동적 프로그램 요구를 처리하는 데 있어서 새로운 프로세스를 만드는 것이 CPU를 사용하는 비율을 높게 만드는 것이다. 이로 인해 CPU의 개수에 따라 성능 차가 생기는 것이다. 또한 웹 서버 사이에서 성능을 비교한다면 전체적으로 TPW의 성능이 가장 높음을 알 수 있었고, RBW와 TBW의 성능은 환경에 따라 다른 결과를 보였다. 쓰레드의 개수가 조절되고, 단계 사이의 큐가 없기 때문에 TPW는 다른 웹 서버들 보다 좋은 성능을 보이는 것이다.

4.6 전체 실험 결과 분석

정적 파일 요구에 대해서 웹 서버가 처리하는 경우에는 CPU 개수에 따른 성능 차이가 없었다. 이는 웹 서버가 정적 파일을 처리하는 데 있어서 CPU를 사용하는 비율이 극히 적기 때문이다. 요구를 분석하는 단계를 제외하고는 CPU를 사용하는 단계가 거의 없다. 요구를 분석하는 단계마저도 CPU를 사용하는 비율이 크지 않기 때문에 CPU 개수에 따른 성능 차가 없는 것이다. 이에 반해 동적 프로그램에 대해서 처리를 하는 경우에는 새로운 프로세스를 하나 만들어서 그 프로그램을 처리하여야 하는데 이 때 새로운 프로세스를 만드는 과정이 CPU를 많이 사용하기 때문에 CPU의 개수에 따른 성능 차가 생기는 것이다.

<표 1>은 웹 서버 사이의 성능을 비교한 표이다. <표 1>에서는 CGI비율과 정적 파일 크기에 따라서 결과를 나타냈지만 이 밖에도 CPU 개수 또한 성능에 영향을 미치게 된다. 하지만 CPU 개수가 하나 이상인 경우에는 항상 TPW, RBW, TBW의 성능 순을 나타내기 때문에 이 표에서는 그 결과를 표시하지 않았다.

표 1 웹 서버의 성능 비교(CPU 개수 = 1개)

CGI비율 / 정적 파일 크기	높음	낮음
115 K byte	TPW>TBW>RBW	TPW>TBW>TBW
50 K byte	TPW>RBW>TBW	TPW>RBW>TBW
500 byte	TPW>TBW>RBW	TPW>TBW>RBW

표를 보면 정적 파일 크기가 500 byte 즉, 정적 파일 크기가 작은 경우이거나 혹은 정적 파일 크기는 115K byte로 크지만 CGI의 비율이 높은 경우에는 TPW, TBW, RBW 의 순으로 전체 성능이 측정이 됨을 알 수 있고 나머지 경우에는 모두 TPW, RBW, TBW 순으로 전체 성능이 나타남을 알 수 있다. TPW의 경우에는 쓰레드의 개수가 조절되어서 병목 현상이 일어나더라도 다른 웹 서버들에 비해 조절이 가능하다. 또한 큐는 요구를 받아들이는 단계에만 존재하고 나머지 단계 사이에서는 존재하지 않으므로 큐에 대한 병목 현상 또한 없어지게 된다. 이로 인해 다른 웹 서버들에 비해 좋은 성능을 나타내는 것이다. 이에 반해 RBW와 TBW는 병목 현상이 일어나는 경우 병목 현상을 조절할 수 없으므로 TPW에 비해 안 좋은 성능을 나타내는 것이다. 특히 RBW는 CPU 개수가 작은 경우에 하나의 CPU에서 많은 쓰레드가 수행이 되는 경우 좋지 않은 성능을 보임을 알 수 있었고, TBW는 단계 사이에서 걸리는 수행 시간에 많은 차이가 보이는 경우 큐의 병목 현상으로 인하여 좋지 않은 성능을 보인다.

5. 결론 및 추후 연구 방향

기존의 웹 서버에 관한 연구들은 보통 하나의 웹 서버 구조, 정적 파일 요구 그리고 하나의 CPU를 가지고 실험을 한 경우가 대부분이다. 본 논문에서는 멀티 쓰레드 기법을 사용한 세 가지 병행 웹 서버 구조 즉 RBW 구조, TBW 구조, TPW 구조 를 구현하였고 다양한 환경에서 실험을 통하여 세 가지 웹 서버를 비교, 분석하였다. RBW 구조는 요구가 서버측으로 들어올 때마다 쓰레드를 생성하는 병행 구조이고, TBW 구조는 서버측의 처리 단계마다 쓰레드를 할당하는 파이프라인 형식의 병행 구조이다. 마지막으로 TPW 구조는 위의 두 병행 구조를 혼합한 형식의 병행 구조이다. 각 구조의 웹 서버는 리눅스 환경에서 Pthread 라이브러리와 Socket 라이브러리를 이용하여 구현하였다.

실험을 통해 각 웹 서버의 성능을 측정함에 있어서 지표로 삼은 것은 초당 HTTP 응답수 즉, 서버의 Throughput이다. 실험 결과에 의하면 TPW가 모든 환경에서 다른 웹 서버들에 비해 좋은 성능을 보였다. TBW와 RBW의 성능은 환경에 따라 서로 다른 결과를 보여주었다. TBW는 CPU의 개수가 작고 요구하는 파일의 크기가 작을 때와 요구하는 파일의 크기가 크지만 CPU의 개수가 작고 요구하는 동적 프로그램의 비율이 큰 경우 RBW 구조보다는 좋은 성능을 보인 데 반해 나머지 환경에서는 RBW 구조가 TBW 구조보다는 좋

은 성능을 보였다. 즉, 서버를 구축함에 있어, 작은 규모의 웹 서버를 구축하고자 하고, 작은 파일이 주를 이룰 때는 TBW가 RBW보다 좋은 성능을 나타내게 된다. 하지만 더 많은 CPU 개수 등 더 나은 웹 서버 환경 구축이 가능한 경우 혹은 큰 파일을 주로 서비스 하는 VOD와 같은 경우에는 RBW가 TBW보다 좋은 성능을 나타내게 되는 것이다. 이와 같은 결과는 TBW 구조에서 이용되는 큐의 병목 현상과 RBW 구조의 많은 쓰레드의 개수에 의한 과부하로 나타난 현상이다. 이와 같은 다양한 CPU 개수와 파라미터를 통한 실험 결과는 다중 처리기 상에서 고성능 웹 서버를 구축하는 데 있어 활용할 수 있다.

본 논문에서는 실험 환경에 따라 TBW의 각 단계마다 할당되는 쓰레드의 개수를 고정된 수로 결정하였으나 환경에 따라 최적의 효율을 내도록 동적으로 쓰레드의 개수를 결정할 수 있는 알고리즘에 대한 연구가 필요하다. 또한 그런 연구를 기초로 각각의 환경에 따라 실험을 통하여 최적의 웹 서버 구조를 찾아낼 수 있을 것이다. 향후 연구에서는 각 웹 서버 구조를 향상시킬 수 있는 방안과 더불어 좀 더 세분화된 환경에서의 실험을 통하여 실제 고성능 웹 서버를 구축하는 데 도움이 될 수 있도록 할 것이다.

참 고 문 헌

[1] Yiming Hu, Ashwini Nanda and Qing Yang, "Measurement, Analysis and Performance Improvement of the Apache Web Server," Proceedings of the 1999 IEEE International Performance, Computing and Communications Conference , 1999.

[2] D. J. Yates, V. Almeida and J. M. Almeida, On the Interaction Between an Operating System and Web Server, Technical Report CS 97-012, Boston University,1997.

[3] Mark E. Crovella, Robert Frangioso and Mor Harchol-Balter, Connection Scheduling in Web Servers, Technical Report CS 99-003, Boston University, 1999.

[4] Apache Group. Apache HTTP Server Project, http://www.apache.org/, 1999.

[5] James E. Pitkow, "A Simple Yet Robust Caching Algorithm based upon Dynamic Access Patterns," Proceedings of the Second International World Wide Web Conference, 1994.

[6] R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk and T. Berners-Lee, RFC 2068 Hypertext Transfer Protocol HTTP/1.1, UC Irvine, Digital Equipment Corporation, MIT, 1997.

[7] N.J. Yeager and R.E. McGrath, Web Server Technology, Morgan Kaufman Publishers, 1996.

[8] M.F. Arlitt and C.L. Williamson, "Web Server Workload Characterization: The Search for Invariants," Proceeding of SIGMETRICS96, 1996.

[9] P. Eriksson, phttpd, http://phttpd.signum.se/phttpd, 1999.

[10] J.C. Hu, S.M. Mungee, and D.C. Schmidt, Principles for Developing and Measuring High-performance Web Servers over ATM, Technical Report WUCS-97-09, Washington University, 1997.

[11] WWW Consortium, Jigsaw HTTP Server, http://www.w3.org/Jigsaw/, 1997.

[12] Arun Iyengar, Ed MacNair and Thao Nguyen, Web Server Performance Under Heavy Loads, Technical Report, IBM Research Division, 1997.

[13] Netscape Inc., Netscape Enterprise Server, http://home.netscape.com/enterprise/v3.6/index.html, 1999.

[14] 제영희, 병행 웹 서버의 설계 및 성능 평가, 서강대학교 석사 학위 논문, 1997.

[15] B. Nichols, D. Buttlar and J.P. Farrell, Pthreads [16]Programming, OReilly & Associates Inc., 1996.

[16] W.R. Stevens, Univ Network Prentice-Hall Press, 1991.

[17] Remy Card, Eric Dumas, Franck Mevel, The Linux Kernel Book, Wiley Press, 1998.

정 진 국

1994년 서강대 전자계산학과 졸업. 2000년 서강대 컴퓨터학과 석사. 2000년 ~ 현재 서강대 컴퓨터학과 박사 과정.

낭 종 호

1986년 서강대 전자계산학과 졸업. 1988년 한국과학기술원 석사. 1992년 한국과학기술원 박사. 1992년 ~ 1993년 Fujitsu연구소 연구원. 1993년 ~ 현재 서강대학교 컴퓨터학과 교수. 현재 서강대학교 컴퓨터학과 부교수

박 성 용

1987년 서강대 전자계산학과 졸업. 1994년 Syracuse 대학 Computer science 석사. 1998년 Syracuse 대학 Computer science 박사. 1998년 ~ 1999년 Bellcore 연구소 연구원. 1999년 ~ 현재 서강대학교 컴퓨터학과 조교수.