학습 기반의 동적 쓰레드 풀 기법을 적용한 웹 서버의 설계 및 구현

(Design and Implementation of a Web Server Using a Learning-based Dynamic Thread Pool Scheme)

유 서 희 † 강 동 현 † 이 권 용 † 박 성 용 ††
(SeoHee Yoo) (DongHyun Kang) (Kwonyong Lee) (Sungyong Park)

요 약 네트워크의 발전에 따라 사용자들이 늘어나게 되면서 웹 서버들은 동시에 접속하는 다수 사용자의 서비스 요청을 처리할 수 있는 다중 쓰레드 기법을 활용하고 있다. 고정된 쓰레드 풀 기법은 고정적인 시스템 자원을 점유해야 하는 문제점이 있다. 반면에 동적으로 쓰레드 풀 기법인 워터마크 쓰레드 풀 기법은 사용자의 요청량에 따라 쓰레드 수를 적절하게 조절하지만, 지정한 최대값을 넘는 요청량에 대해서는 응답이 제때에 이루어지지 않는 단점이 있다. 따라서 본 논문에서는 다양한 요청량이 존재하는 다중 쓰레드 환경의 서버 프로그래밍을 위한 학습 기반의 동적 쓰레드 풀 기법을 적용한 웹 서버를 제안한다. 제안하는 기법은 쓰레드 풀을 사용하는 웹 서버 중 아파치(Apache) worker 다중 처리 모듈(Multi-processing Module)에 AR(Auto Regressive) 기법을 통해 다음 주기의 작업 요청량을 예측하고 사전에 쓰레드를 생성한다. 기존 기법과 달리, 일정주기의 증감 추세가 없는 작업 요청량에도 필요한 쓰레드의 수를 정확하게 설정하기 위해 최근접 이웃(K-Nearest Neighbor) 알고리즘을 사용하여 작업 요청량에 따른 쓰레드의 수를 사전에 학습한다. 필요한 쓰레드의 수를 설정하기 위해 사전에 학습 되어진 개체들과 비교하여 유사한 개체를 선택하여 예측된 작업 요청량에 따른 쓰레드의 수를 결정하고 쓰레드를 생성한다. 본 논문에서는 필요한 쓰레드의 수를 동적으로 변경함으로써 사용자 응답 시간을 빠르게 하고, 사용자의 요청량에 맞게 쓰레드 수를 관리함으로써 시스템 자원의 활용도를 높일 수 있다.

키워드 : 쓰레드 풀, 다중 처리 모듈, AR, 최근접 이웃 알고리즘

Abstract As the number of user increases according to the improvement of the network, the multi-thread schemes are used to process the service requests of several users who are connected simultaneously. The static thread pool scheme has the problem of occupying a static amount of system resources. On the other hand, the dynamic thread pool scheme can control the number of threads according to the users' requests. However, it has disadvantage that this scheme cannot react to the requests which are larger than the maximum value assigned. In this paper, a web server using a learning-based dynamic thread pool scheme is suggested, which will be running on a server programming of a multi-thread environment. The suggested scheme adds the creation of the threads through the prediction of the next number of periodic requests using Auto Regressive scheme with the web server apache worker MPM (Multi-processing Module). Unlike previous schemes, in order to set the exact number of the necessary threads during the unchanged number of work requests in a certain period, K-Nearest Neighbor algorithm is used to learn the number of threads in advance

· 본 연구는 지식경제부 및 한국산업기술평가관리원의 IT산업원천기술개발사업의 일환으로 수행하였음[2009-KI002090, 신뢰성 컴퓨팅(Trustworthy Computing) 기반 기술 개발]

논문접수 : 2009년 5월 4일
심사완료 : 2009년 10월 21일

† 학생회원 : 서강대학교 컴퓨터공학과
rthlove@sogang.ac.kr
donghyuny@sogang.ac.kr
dlrnjsdyd@sogang.ac.kr
†† 종신회원 : 서강대학교 컴퓨터공학과 교수
parksy@sogang.ac.kr

Copyright©2010 한국정보과학회 : 개인 목적이나 교육 목적인 경우, 이 저작물의 전체 또는 일부에 대한 복사본 혹은 디지털 사본의 제작을 허가합니다. 이 때, 사본은 상업적 수단으로 사용할 수 없으며 첫 페이지에 본 문구와 출처를 반드시 명시해야 합니다. 이 외의 목적으로 복제, 배포, 출판, 전송 등 모든 유형의 사용행위를 하는 경우에 대하여는 사전에 허가를 얻고 비용을 지불해야 합니다.

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제1호(2010.1)

according to the number of requests. The required number of threads is set by comparing with the previously learned objects. Then, the similar objects are selected to decide the number of the threads according to the request, and they create the threads. In this paper, the response time has decreased by modifying the number of threads dynamically, and the system resources can be used more efficiently by managing the number of threads according to the requests.

Key words : Thread pool, multi-processing module, AR, K-Nearest Neighbor algorithm

1. 서 론

최근 들어 네트워크의 급속한 발전과 웹의 등장으로 서버-클라이언트 방식의 서비스가 많아짐에 따라 다수 사용자들의 동시 접속을 처리하기 위한 능력이 필요하게 되었다. 초기에는 다중 프로세스 기법을 사용하여 왔지만, 시스템의 전체 사용자 수가 증가하면서 새로운 프로세스의 생성이나 프로세스간의 문맥 전환 등으로 인해 과부하가 큰 부담이 되고 있어 오늘날에는 대부분 다중 쓰레드 기법을 활용하고 있다. 그러나 쓰레드의 생성과 제거에는 오버헤드가 발생하고 이는 서비스 응답 시간 지연 등의 웹서버 성능저하를 가져오게 된다. 따라서 미리 쓰레드 풀(Thread pool) [1]을 생성해두고 이를 사용하여 서비스 요청을 처리하는 기법이 사용된다.

쓰레드 풀은 지정된 수만큼 쓰레드를 미리 생성하여 하나의 풀을 형성시켜 대기 중인 쓰레드로 다수의 사용자들이 요청하는 서비스를 동시에 처리해준다. 사전에 쓰레드를 생성해 놓음으로써 서비스 요청시 쓰레드 생성으로 인한 응답 시간 지연을 방지할 수 있고, 생성되었던 쓰레드를 재사용하여 쓰레드 생성과 삭제로 인한 과부하를 감소시킬 수 있다. 그러나 쓰레드 풀을 형성함으로써 사용자의 서비스 요청이 적은 경우 유휴 쓰레드가 발생하게 되고 이는 시스템 자원의 낭비를 의미한다. 또한 쓰레드 풀 이상의 서비스 요청에 대해서는 할당 가능한 쓰레드가 생길 때까지 서비스 지연이 발생하는 문제점이 있다. 따라서 쓰레드 풀의 쓰레드 수를 적절하게 조절할 수 있다면 시스템 자원 낭비의 문제를 막을 수 있다. 이에 따라 쓰레드 풀 크기를 동적으로 변화시키는 여러 연구가 진행되고 있다[2-4]. 기존 동적 쓰레드 풀 기법에서는 일정 주기의 증감 추이를 바탕으로 필요한 쓰레드의 수를 결정하기 때문에 일정 구간의 증감 패턴을 가지지 않는 서비스 요청량에 대해서는 정확한 예측이 어렵다. 실제 웹 서버에서 발생되는 서비스 요청량은 일정 주기 동안 계속적으로 증가하거나 감소하는 패턴만을 가지지 않는다. 따라서 특정한 증감 패턴을 가지지 않는 작업 요청량에 대해서도 필요한 쓰레드의 수를 예측하여 쓰레드 풀의 크기를 동적으로 조절한다면, 서비스 지연 없이 효율적으로 시스템 자원을 사용하여 웹 서비스를 제공할 수 있게 될 것이다.

본 논문에서 제안하는 학습 기반의 동적 쓰레드 풀 기법이 적용된 웹 서버는 사용가능한 시스템 자원이 한정되어 있는 환경에서 효율적인 자원 관리를 위해 다수 사용자들의 불규칙적인 서비스 요청이 특정 증감패턴 없이 발생하는 경우의 서비스 요청량을 기반으로 성능을 평가한다. 결과적으로 본 논문에서 제안한 학습기반의 동적 쓰레드 풀 기법을 통해 예측된 쓰레드 수에 따라 쓰레드 풀의 크기를 조절하여 최소한의 시스템 자원으로 서비스 지연 없는 웹 서비스를 제공한다.

본 논문의 구성은 다음과 같다. 1장에서는 연구의 배경 및 연구의 방법에 대해 살펴보고, 2장에서는 기존의 연구들과 이들의 문제점에 대해 살펴본다. 3장에서는 제안하는 기법의 환경에 대해서 분석하고, 4장에서는 아파치(Apache) worker MPM에 제안한 기법을 적용하여 구현한다. 5장에서는 성능 평가와 이에 대한 분석을 수행하며, 6장에서는 본 논문의 결론과 향후 과제에 대해 논의한다.

2. 관련 연구

본 장에서는 기존 쓰레드 기법을 살펴보고, 이러한 기존 연구의 문제점에 대해 분석한다. 요구 기반 쓰레드 기법[5]은 사용자의 요청을 받아 처리할 워커 쓰레드를 생성하고 사용자의 요청을 넘겨주는 역할을 하는 입출력 쓰레드와 입출력 쓰레드에 의해 생성되어 사용자의 요청을 처리하고 소멸하는 워커 쓰레드로 구성된다. 이 기법은 사용자의 요청이 있을 때마다 새로운 쓰레드를 생성하여 해당 요청을 처리하고 쓰레드를 삭제해야 하므로 쓰레드 생성으로 인한 응답속도의 저하 및 계속되는 쓰레드 생성 및 삭제로 인한 과부하로 사용자의 요청이 많은 경우에는 비효율적인 구조라 할 수 있다.

이러한 요구 기반 쓰레드 기법의 문제점을 개선하기 위해 워커 쓰레드 풀 기법[3]이 제시되었다. 이 기법은 서버 시작 시 일정한 개수의 워커 쓰레드를 미리 생성시키는 방법으로, 쓰레드의 생성 및 삭제에 따른 과부하를 없앰으로써 응답시간을 단축하고, 과도한 쓰레드 생성으로 인한 시스템 자원 고갈의 위험을 없앤다. 그러나 워커 쓰레드의 개수를 고정하기 때문에 사용자의 요청이 적은 경우 일정 개수의 워커 쓰레드를 유지해야 하므로 일정한 시스템 자원을 할당해야 한다. 또 사용자의 요청이 많은 경우 할당 가능한 시스템 자원이 남아 있

더라도 사용자의 요청이 워커 쓰레드가 앞의 일을 마치기를 대기하고 있어야 하기 때문에 시스템의 자원을 효율적으로 활용할 수 없다. 시스템의 자원을 보다 효율적으로 활용하기 위해, 워터마크 쓰레드 풀 기법은 쓰레드 풀의 크기를 요청 양에 따라 동적으로 변하게 한다. 이를 위해 워커 쓰레드 풀에 낮은 워터마크와 높은 워터마크를 설정하고, 서비스 시작 시 낮은 워터마크만큼의 워커 쓰레드를 미리 생성한다. 사용자의 요청이 많아 낮은 워터마크만큼의 쓰레드가 다 사용되면 높은 워터마크 이전까지 워커 쓰레드를 생성한다. 워커 쓰레드의 개수가 높은 워터마크에 이르면 더 이상 쓰레드를 생성하지 않는다. 이 기법은 쓰레드 풀의 크기를 조절할 수 있는 장점이 있지만, 높은 워터마크를 넘어가도록 요청량이 올 경우 요청에 대한 응답이 제때에 이루어지지 않는다는 단점이 있다. 다양한 요청량에 대한 해결책으로, 지수 평균 기법[6]은 과거의 값들을 근거로 다음에는 어떠한 값이 오게 될 것인지를 예측하기 위한 방법으로, 서비스 요청량의 증감량을 파악해서 필요한 쓰레드를 예측할 수 있다. 그러나 이 기법은 사용자의 요청량이 증가하는 구간에서 실제로 예측되는 값은 현재값 보다 적은 값이 나오게 되어 현재의 증가하는 추세를 전혀 반영하지 못하는 단점이 존재한다.

이러한 문제점을 보완하기 위해서 지수 평균식을 수정하여 다음 주기에 필요한 쓰레드의 수를 계산한다[4]. 이는 전 단계와 현재 값의 차이를 통해 서비스 요청량의 증감 여부를 판별하여 다음 주기에 필요한 쓰레드의 수를 계산하는 방법이다. 쓰레드의 양이 증가하는 추세이면 현재 쓰레드 개수와 예측치의 차이를 현재의 쓰레드 개수에 더한 값을 반영하고, 쓰레드의 양이 감소하는 추세이면 사용자의 요청량보다 항상 많은 수의 쓰레드를 유지하고 있기 때문에 지수 평균을 그대로 적용한다. 이는 서비스 요청량의 증감의 추세가 정확하게 파악이 된다면 과거의 값들을 기반으로 현재 필요한 쓰레드의 양을 계산하여 응답 시간을 최소화할 수 있다. 그러나 서비스 요청량 증감 추세가 정확하게 파악되지 않는다면 다음 주기에 필요한 쓰레드 양을 정확하게 판단하기 어려운 단점이 존재한다.

기존의 다양한 쓰레드 풀 기법 중 고정적인 쓰레드 풀을 사용하는 기법은 최적의 쓰레드의 개수를 유지하면서 다수의 요청에 빠르게 응답할 수 있는 장점이 있지만 사용자의 접속이 적을 때에도 고정된 시스템 자원을 점유하여야 하는 자원의 효율적 사용에 문제가 있었다. 그리고 동적으로 쓰레드 풀 크기를 조절하는 기법 중, 예측 기반의 동적 쓰레드 풀 기법은 사용자 요청량의 증감 추세를 판단하여 필요한 쓰레드의 수를 설정한다. 그러나 이러한 기법은 일정 주기의 증감 추세가 없는 사용자 요청량에 대해서는 필요한 쓰레드의 수를 정확하게 예측하기가 어렵고, 서비스 요청량 증감 추세를 정확히 맞추기 어려운 경우에도 사용하기가 어렵다. 또한, 서비스 요청량이 불규칙하거나 크기의 편차가 큰 경우, 그리고 동시에 갑작스럽게 요청량이 많아지는 경우에는 예측의 정확성이 떨어지게 되어 추가적인 쓰레드 생성에 따른 과부하 발생으로 사용자 응답 시간이 늦춰지게 된다. 따라서 일정 주기 동안의 서비스 요청량의 증감 패턴을 가지지 않는 경우에서도 예측의 정확성을 높임으로써 효율적인 시스템 자원의 사용과 함께 사용자 응답 시간의 최소화가 중요하게 된다.

3. 아파치(Apache) 웹 서버의 분석

본 장에서는 제안하는 기법의 구현환경인 아파치[7] 웹 서버의 구성을 설명하고, 이를 구성하는 다중 처리 모듈(Multi-Processing Module, MPM)에 종류와 특징을 살펴본다. 또 여러 모듈들 중, 본 논문에서 제안하는 알고리즘이 적용된 worker 다중 처리 모듈에 대해 분석한다. 아파치웹 서버는 APR(Apache Portable Runtime)의 기능으로 좀 더 많은 플랫폼에서 쉽게 포팅될 수 있고, 더불어 관리까지 쉽게 이루어질 수 있도록 도와준다. 또한, 모듈화된 설계로 다양한 환경의 다양한 플랫폼에서 동작할 수 있도록 설계되었다. 서버는 시스템의 네트워크 포트에 연결하고, 요청을 받아들이며, 받아들인 요청을 처리하기 위해 자식 프로세스 또는 쓰레드들에게 분배하는 다중 처리 모듈을 선택할 수 있다. 이를 통하여 아파치 웹 서버의 확장성을 상당히 증가시킬 수 있고, 각각의 모든 플랫폼에서 이전 버전보다 안정적이고 빠른 속도를 얻을 수 있으며, 개발자들은 코어를 수정해야하는 번거로움 없이 특정 모듈만을 수정하여 각 운영체제에 맞게 가장 효율적인 구현을 할 수 있다.

3.1 다양한 다중 처리 모듈의 종류

다중 처리 모듈은 아파치 2.x가 모든 플랫폼에서 완벽한 멀티 쓰레드 서버로서 동작할 수 있게끔 할 뿐만 아니라, 특정 서버 환경에서 가장 적절한 프로세스 기법을 선택하여 사용할 수 있도록 한다. 따라서 아파치 웹 서버는 받아들인 요청을 처리하기 위해 자식들에게 분배하는 다중처리 모듈을 선택한다. Prefork 다중 처리 모듈은 미리 프로세스를 생성해 놓고 그 프로세스들을 풀에 넣어 두었다가 사용하는 방식이다. 서비스가 시작되어 생성되는 아파치 프로세스가 부모 프로세스이고, 이는 분기하여 자식 프로세스를 한 개 이상 생성한다. 실제 HTTP 서비스를 담당하는 것은 자식 프로세스이고, 이를 제어하는 것은 부모 프로세스이다. 그러므로 이 모듈은 쓰레드가 한 개인 자식 프로세스를 여러 개 사용한다. 각 프로세스는 한 번에 한 연결을 담당하고,

서비스 요청이 많게 되면 프로세스의 수가 증가하게 된다. 그 다음으로, perchild 다중 처리 모듈은 프로세스들의 정적인 풀을 관리하는데, 각 프로세스는 동적으로 변동 가능한 쓰레드들을 관리한다. 다른 다중 처리 모듈들과는 달리, perchild는 프로세스들이 다른 서버 설정(가상 호스트)들과 연관된다. 하나의 프로세스를 특정 가상 호스트와 연관시키게 되면, perchild는 각 가상 호스트가 자신만의 쓰레드 풀을 관리할 수 있도록 허용하여, 그 가상 호스트의 요청에 따라 쓰레드 풀을 증가시키거나 감소시키게 된다. 마지막으로, worker 다중 처리 모듈이 있다. 이것은 프로세스의 동적 풀을 관리하고, 각 프로세스는 고정된 개수의 다중 쓰레드들을 포함한다. 부모 프로세스는 제어를 담당하고, 자식 프로세스에 속한 각 쓰레드는 한 번에 한 연결을 담당한다. 그리고 사용자의 요청을 처리하는데 있어 설정된 한계를 초과하게 되는 경우에는 새로운 자식 프로세스를 생성하게 된다. 이 모듈의 자식 프로세스는 크게 3개로 구분할 수 있다. starter는 프로세스내의 다른 쓰레드를 생성하고 초기화하는 쓰레드이고, listener는 다른 프로세스의 listener와 accept mutex에 참여하는 쓰레드이다. 각각의 자식 프로세스에는 하나의 listener가 있는데 listener 중에 하나가 소켓을 accept mutex에 의해 얻게 되면 자신과 같은 프로세스 내에 있는 worker 쓰레드에게 연결을 넘긴다. 그리고 worker 쓰레드는 실제 서비스를 처리한다. 따라서 본 논문에서는 다중 쓰레드를 활용하여 서비스 요청을 처리하기 위해서 worker 다중 처리 모듈을 기반으로 제안된 알고리즘을 구현한다.

3.2 아파치 웹 서버 worker 다중 처리 모듈의 분석

Worker 다중 처리 모듈은 멀티 프로세스와 멀티 쓰레드의 혼합형 동작 방식을 사용한다. 하나의 부모 프로세스와 다수의 자식 프로세스가 존재하는데, 각각의 자식 프로세스는 다수의 쓰레드를 포함하여, 실제적인 서비스 처리는 각 자식 프로세스의 쓰레드가 처리한다. 아파치 서비스가 시작되면 부모 프로세스는 초기화 작업으로 환경 설정 파일에 설정되어 있는 지시자들을 통해 자식 프로세스와 쓰레드들을 생성한다. 본 논문에서는 아파치 worker 다중 처리 모듈 환경 설정 파일에 설정되어 있는 기본값들을 사용하였으며 그림 1과 같다.

Worker 다중처리 모듈 서비스는 그림 2와 같이 진행

StartServer	2
MaxClients	150
MinSpareThreads	25
MaxSpareThreads	75
ThreadsPerChild	25
MaxRequestsPerChild	0

그림 1 worker 다중 처리 모듈 지시어 기본 설정값

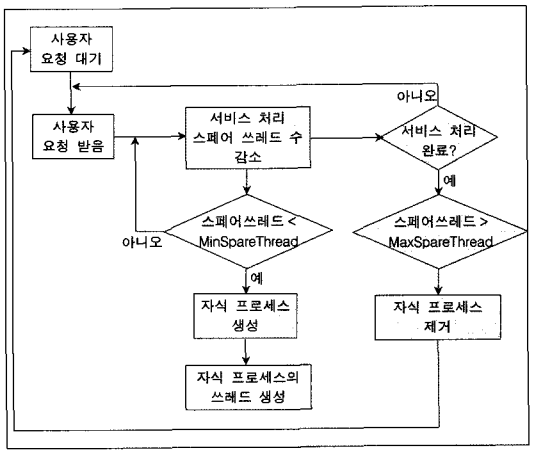


그림 2 worker 다중 처리 모듈 서비스 처리 순서도

된다. 아파치 웹 서버의 초기화 작업이 끝나면 ap_mpm_run 함수를 통해 부모 프로세스와 자식 프로세스로 나누어지기 시작하며, 이 때, 부모 프로세스는 StartServers에 설정되어 있는 수만큼의 자식 프로세스들을 생성하고, 각 자식 프로세스들은 쓰레드를 생성한다. 자식 프로세스의 starter 쓰레드는 ThreadsPerChild의 설정에 따라 worker 쓰레드를 생성한다.

현재 생성된 쓰레드들 중 요청에 대한 서비스를 하지 않고 있는 쓰레드를 스페어(spare) 쓰레드라고 하며, 프로그램 시작시 어떠한 요청도 없다면 StartServer * ThreadsPerChild가 스페어 쓰레드 수가 된다. 사용자의 서비스 요청이 들어와서 서비스 처리를 하게 되면 스페어 쓰레드 수가 줄어들게 되는데, 부모 프로세스는 이를 감시하고 있다가 스페어 쓰레드의 수가 MinSpareThread보다 적어지게 되면 자식 프로세스를 추가로 생성한다. 그리고 새로 생성된 자식 프로세스는 ThreadsPerChild에 설정되어 있는 만큼 쓰레드를 생성한다. 이러한 방식으로 생성될 수 있는 자식 프로세스의 개수는 worker 다중 처리 모듈 파일에서 설정된 ServerLimit까지 가능하다. ServerLimit는 최대 생성 가능한 자식 프로세스의 수를 의미하고, 16을 기본값으로 설정되어 있다. 서비스 요청이 모두 끝나서 스페어 쓰레드의 수가 설정된 MaxSpareThread보다 많게 되면, 부모 프로세스는 자식 프로세스 중 하나를 제거한다. 이렇게 해서 스페어 쓰레드의 수는 MinSpareThread와 MaxSpareThread사이의 범위로 유지하게 된다. MaxClients는 동시에 서비스 가능한 사용자의 수를 설정하는데 쓰이고, 이는 ServerLimit * ThreadsPerChild까지 제한된다. 만약 동시에 MaxClients 수보다 많이 들어오면 큐(queue)에서 대기하게 된다. MaxRequestsPerChild는 자식 프로세스가 처리 가능한 서비스 양을 의미하며, 자식 프로

세스가 처리한 요청의 수가 설정된 수만큼 되면 그 자식 프로세스를 제거한다. 만약, 이 지시자가 0으로 설정되어 있다면 처리한 요청의 수를 제한하지 않는다.

4. 학습 기반의 동적 쓰레드 풀 기법 구현

본 장에서는 학습 기반의 동적 쓰레드 풀의 설계 목표에 대해 알아본다. 아파치 웹 서버의 worker 다중 처리 모듈을 수정한 쓰레드 풀에 AR기법을 통해 사용자의 요청량을 예측하고, 최근접 이웃 알고리즘을 통해 학습된 개체들과 비교하여 쓰레드의 개수를 결정하는 방법에 대하여 설명한다. 또, 학습기반의 동적 쓰레드 풀 기법을 worker 다중 처리 모듈에 어떻게 적용시켰는지 설명하고, 이를 위해 사용된 감시자 쓰레드의 역할에 대해 알아본다.

4.1 제안된 동적 쓰레드 풀 기법의 설계 목표 및 전체 구조

본 논문은 동시에 다수 사용자들의 서비스 요청에 대해 빠른 응답을 위해 사전에 쓰레드를 생성해 놓음으로써 요청에 대한 즉시 처리가 가능한 쓰레드 풀 구조를 이용한다. 기존에 존재하는 다양한 쓰레드 풀 구조 중에서 워터 마크 쓰레드 풀을 기반으로 하여 서비스의 요청량을 사전에 예측하고 쓰레드를 미리 생성함으로써 시스템의 자원을 효율적으로 사용하게 된다. 사전에 쓰레드를 생성해 두어 서비스 요청시 쓰레드 생성에 필요한 오버헤드를 줄이고, 사용자 요청이 즉시 처리되도록 응답 시간 지연을 방지할 수 있도록 한다. 또한 필요한 쓰레드 수를 예측하는데 있어 정확성을 높여 필요 이상의 쓰레드를 생성을 막아 시스템 자원 사용을 최적화하는 것을 목표로 한다.

본 논문에서 제안하는 학습기반 동적 쓰레드 풀 기법의 기법의 전체적인 흐름은 그림 3과 같다.

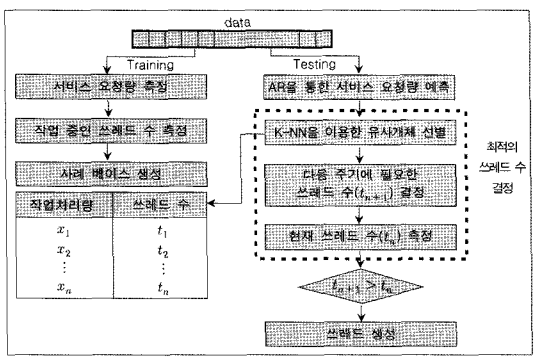


그림 3 학습 기반의 동적 쓰레드 풀 기법

4.2 서비스 요청량 예측

본 논문에서는 일반적으로 예측기법에 널리 쓰이는 대표적인 시계열 분석법인 AR기법[8]을 사용하였다. 이 기법을 통해, 현재까지 수집된 데이터를 분석하고, 다음 주기의 사용자 요청량을 예측한다. 다음 주기에 필요한 쓰레드의 수를 결정하는데, 각 주기간의 오차의 편차를 줄이기 위해 Yule-Walker식을 기반으로 하는 최소자승법을 이용하여 AR계수를 결정하였다. 예측의 정확성을 높이기 위해 AR 계수를 계산하고, AR기법을 사용하기 위해서는 우선 AR의 차수를 결정해야할 필요가 잇다. 이를 위하여 각 차수별 AR예측의 오차를 측정한 결과를 그림 4에서 얻을 수 있었다. 이 그래프를 통해 차수가 증가함에 따라 제곱 평균 오차가 줄어들어 예측에 대한 정확성이 높아지는 것을 알 수 있다. 그러나 order가 8이상일 경우, 이들 오차 간의 큰 차이가 없고, order가 증가함에 따라 AR계수를 계산하는 시간이 증가하게 된다. Order의 값을 너무 높게 설정하면, 제곱 평균 오차가 감소하지만 이를 위해 사전의 많은 데이터를 필요로 한다. 그러나 order의 값을 너무 낮게 설정하면, 예측의 정확성이 떨어진다. 그림 4를 보면, 오차율이 급격하게 낮아지는 첫 지점은 8이다. 따라서 본 논문에서는 예측의 오차율을 최소화하고 예측값을 계산하는 시간을 최소화하기 위하여 order를 8로 결정하였다.

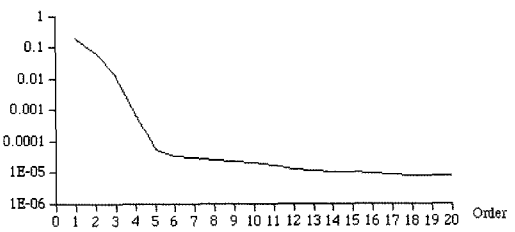


그림 4 제곱 평균 오차율

사용자 서비스 요청을 응답시간의 지연 없이 처리할 수 있는 최적의 쓰레드 수를 결정하기 위하여 실시간으로 들어오는 서비스 요청량에 따라 1초 간격으로 현재 사용자에 의해 요청된 서비스의 양과 현재 작업 중인 쓰레드의 개수를 계산하여 사례 베이스에 저장하여 훈련 데이터 집합을 유지한다. 사례 베이스에서 중복되는 서비스 요청량이 있는 경우에는 업데이트하여 항상 최신의 데이터를 유지한다.

4.3 최적의 쓰레드 결정 및 생성

훈련 데이터 집합(training data set)은 실시간으로 들어오는 서비스 요청량에 따라 사용되어진 쓰레드의 수를 저장해서 유지하는 사례 베이스를 구성하고, 이 데이터 집합은 일반적으로 웹 서버에 요청하는 사용자의 요청량[11]을 기반으로 구성하였다. 이러한 사례 베이스로부터 서비스 요청량에 따른 필요한 쓰레드의 수를 결

정하기 위하여 방법으로 최근접 이웃 알고리즘(K-Nearest Neighbor) [9]을 사용한다. 이를 적용하여 다음 주기에 필요한 쓰레드의 수를 결정하여 유사 개체 k개를 선택하기 위한 최적의 k값을 결정해야 할 필요가 있다. 그림 5의 k값의 변화에 따른 정확도의 변화를 관찰한 것이다[10]. 이 그래프에서 볼 수 있듯이 k의 값이 너무 작거나 너무 크면 정확도가 상대적으로 감소하는 것을 알 수 있다. 따라서 본 논문에서는 k값에 따른 정확도 실험을 한 논문을[10] 참조하여 k값을 설정하였고, k값은 가장 정확도가 높은 결과로 나타난 4로 결정하였다.

사례 베이스에서 이전에 같은 서비스 요청량이 있었는지 검색하여 만약 이전에 같은 값이 있었다면 해당 값에 따른 쓰레드의 수를 다음 쓰레드 수로 결정하게 되고, 이전에 같은 값이 없었다면 최근접 이웃 알고리즘을 사용하게 된다. 사례베이스에 저장되어 있는 서비스 요청량들 중 앞서 서비스 요청량 예측을 통해 얻은 예측 서비스 요청량과 가장 유사한 개체 4개를 선별하고 서비스 요청량에 따른 각 쓰레드 수에 대해 평균값을 구해 이를 필요한 쓰레드의 수로 결정한다. 마지막으로 설정된 쓰레드 수를 현재 쓰레드 수와 비교하여 현재 쓰레드의 수보다 더 많은 수가 필요하다면 필요한 수를 계산하여 추가적으로 쓰레드를 생성한다.

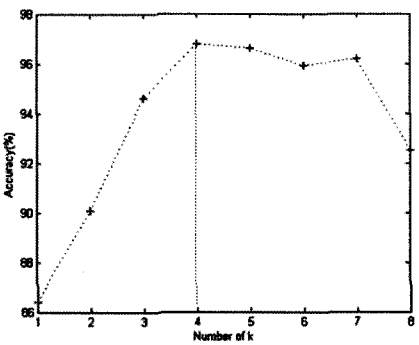


그림 5 k값에 따른 정확도의 변화

4.4 학습 기반의 동적 쓰레드 풀 기법 적용

본문에서 제안하는 쓰레드 풀은 아파치 2.2.10 버전의 worker 다중 처리 모듈을 기반으로 한다. 프로그램이 시작되면 부모 프로세스는 감시자 쓰레드를 생성한 후, 환경 설정 파일에 설정된 지시자들을 이용하여 자식 프로세스와 쓰레드를 생성한다. 자식 프로세스의 listener 쓰레드에 의해 사용자의 요청을 받아서, 사용자의 요청을 처리하는 worker 쓰레드에 전달하여 사용자의 요청을 처리한다. 전체적인 프로그램 구조는 그림 6과 같다.

기본 아파치 웹 서버 worker 다중 처리 모듈에 감시자 쓰레드를 추가하여 현재의 쓰레드 개수를 측정하고

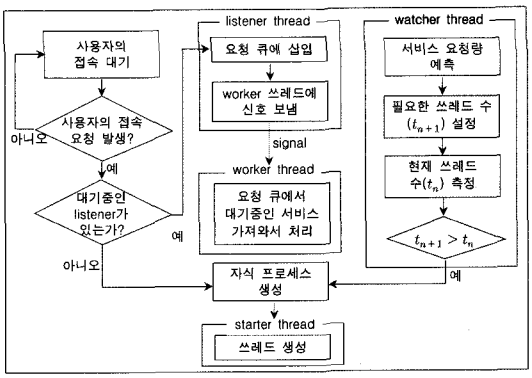


그림 6 프로그램 구조도

서비스 요청량에 필요한 쓰레드의 개수를 사례 베이스에 저장한다. 그 값을 근거로 다음 주기의 서비스 요청량을 예측하여 필요한 쓰레드의 수를 최근접 이웃 알고리즘을 통해 결정하고, 예측된 쓰레드의 수가 현재의 쓰레드 수보다 많으면 새로운 자식 프로세스를 생성함으로써 쓰레드를 추가적으로 생성한다.

서비스 요청에 의해 소켓 연결이 이루어지면 listener 쓰레드에서 감지하여 요청된 서비스를 요청 큐에 삽입하고, 대기 중인 worker 쓰레드에게 서비스 요청을 알리고, worker 쓰레드는 요청 큐에서 대기 중인 서비스를 꺼내서 처리하도록 한다. 만약 대기 중인 worker 쓰레드가 없을 경우에는 새로운 자식 프로세스를 생성하고 쓰레드를 생성하여 대기 중인 요청된 서비스를 처리한다. 요청된 서비스가 모두 처리 되면, 부모 프로세스는 스페어 쓰레드의 수를 최소 스페어 한계치와 최대 스페어 한계치 사이의 수로 유지하기 위해 초과되는 만큼의 자식 프로세스를 하나씩 제거하여 현재 스페어 쓰레드의 수를 일정 수위로 유지한다.

4.5 감시자 쓰레드

본 논문에서 제안하는 기법은 감시자 쓰레드를 통해 동작하며, 이는 아파치 웹 서버가 시작된 후 부모 프로세스와 자식프로세스가 분기되어지는 ap_mpm_run 함수에서 생성된다. 그림 7은 ap_mpm_run 함수에서 감시자 쓰레드를 생성하는 구현 부분이며, 그림 8은 제안하는 기법을 아파치 웹 서버 worker 다중 처리 모듈에 적용하여 구현한 부분이다. 감시자 쓰레드는 1초 간격으로 현재의 서비스 요청량과 현재 작업중인 쓰레드의 개수(work_thread_cnt)를 계산하여 사례 베이스에 저장하고, AR 기법을 이용하여 다음 서비스 요청량(next_workload)을 예측하여 최근접 이웃 알고리즘을 통해 다음 주기에 필요한 쓰레드의 수(predict_thread_cnt)를 결정하게 된다. 그림 9는 AR 기법을 이용하여 다음 주기의 서비스 요청량을 예측하기 위해 구현된 부분이다.

```
int ap_mpm_run ( ... ) {
    ..........
    apr_thread_create ( &thread_read, thread_read_attr,
                        watcher_thread, (void*)NULL, pwatcher )
    server_main_loop ( ... )          // 자식 프로세스 생성
    ..........
```

그림 7 감시자 쓰레드 생성

```
APR_THREAD_FUNC watcher_thread ( ) {

  if (sec == after_sec) {              // 1초 마다
    // 현재 생성된 쓰레드의 개수
    cur_thread_cnt = childCnt * ap_threads_per_child;
    // 현재 작업 중인 쓰레드의 개수
    work_thread_cnt = cur_thread_cnt - idle_thread_cnt;

    공유 메모리를 통해 전체 서비스 요청량(request_sum)을 읽음.
    사례 베이스에 처리된 서비스 양과 사용된 쓰레드 수를 저장.

    next_workload = predictor();     // 다음 서비스 요청량을 예측함
    // 예측된 서비스 요청량에 따라 필요한 쓰레드 수 설정
    predict_thread_cnt = knn_find_required_thread_count();

    if (cur_thread_cnt < predict_thread_cnt) {
        required_child = predict_thread_cnt / ap_threads_per_child;
        for (i=0; i<required_child; i++)
            make_child();            // 자식 프로세스 생성
    }
  }
}
```

그림 8 감시자 쓰레드의 자료 구조

```
int predictor() {
  AutoRegression(inputseries, length, order, coefficients);
  for (i=0; i<length; i++) {
    if (i > order) {
      for (j=0; j<order; j++)
        // AR 계수(coefficients)를 이용하여 다음 값 예측
        estimation += coefficients[j] * series[i-j-1];
    }
  }
  return estimation;
}
```

그림 9 AR 기법의 자료 구조

length는 서비스 요청량으로 입력된 데이터들의 전체 수를 의미하고, order는 AR 계수를 구하기 위해 쓰여지는 값이다.

그림 10은 최근접 이웃 알고리즘을 이용하여 필요한 쓰레드의 수를 결정하기 위해 구현된 부분이며, 유클리디언 거리 공식을 통해 유사 개체를 선별하여 필요한 쓰레드의 수를 결정한다. 설정된 쓰레드 수가 현재의 쓰레드 수보다 더 많이 필요하면 각 자식 프로세스가 생성 가능한 쓰레드의 수를 기반으로 추가로 생성할 자식 프로세스의 수를 계산하여 자식 프로세스를 생성하고 각각의 자식 프로세스는 쓰레드를 생성한다.

전체적인 서비스 요청량의 측정은 그림 11과 같이 각 자식 프로세스의 listener 쓰레드에서 서비스 요청이 큐에 삽입되는 부분에서 서비스 요청량을 누적함으로써 계산한다. 이 값은 감시자 쓰레드가 공유 메모리를 통해 각

```
int knn_find_required_thread_count() {
   유클리디언 거리 공식을 통해 가장 유사한 개체 4개 선별
   if ( distance == 0 )
      return 해당 서비스 요청량에 따른 쓰레드 수;
   else
      return 선별되어진 유사 개체 4개의 쓰레드 수 평균 값;
}
```

그림 10 최근접 이웃 알고리즘의 자료구조

```
static void *listener_thread ( ... ) {
            .....
        rs = apr_global_mutex_lock (exipc_mutex);
        if (APR_SUCCESS == rs) {
            base = (exipc_data *) apr_shm_baseaddr_get (exipc_shm);
            base -> request_sum++;     // 요청된 서비스의 양 누적
            apr_global_mutex_unlock (exipc_mutex);
        } else {    ap_log_error( ... );   }
        ap_queue_push (worker_queue, csd, ptrans);
            .....
}
```

그림 11 전체 서비스 요청량 측정 방법

자식 프로세스의 쓰레드들이 처리할 서비스 전체 요청량을 합산하여 전체적인 서비스 요청량의 합을 계산한다. 공유메모리의 구현은 APR 라이브러리에서 제공되는 ap_shm을 이용하였다.

5. 성능 평가

본 장에서는 앞서 제안한 효율적인 자원 관리를 위한 학습 기반의 동적인 쓰레드 풀 기법을 이용하여 성능을 측정하였다.

5.1 성능 평가 개요

성능 평가의 비교 대상으로는 지수 평균 기법을 이용한 동적 쓰레드 풀 기법, 워터 마크 쓰레드 풀, 그리고 워커 쓰레드 풀로 선정하였다. 첫 번째 비교 대상은 사용자의 요구에 대한 응답 시간이 얼마만큼의 개선이 이루어졌는지를 측정하기 위해서 선택하였고, 두 번째 비교 대상은 낮은 워터마크와 높은 워터마크를 설정하여 운영하는 기법에 예측 기법을 넣음으로써 사용자의 응답 시간의 변화를 비교하기 위해 선택하였다. 그리고 세 번째 비교 대상은 제안한 기법과의 메모리 효율성에 대해 비교하기 위해 선택하였다.

5.1.1 성능 시험을 위한 환경 구성

본 실험에서 사용된 환경은 쓰레드 풀 기법을 구현한 서버가 동작하게 될 서버 시스템과 서버에 서비스 요청을 한 후 응답 시간 측정을 위한 클라이언트 시스템으로 구성되어 있다.

아파치 웹 서버 2.2.10 버전을 기반으로 리눅스 환경에서 APR 라이브러리와 C언어를 이용하여 구현하였다.

5.1.2 성능 측정 방법

그림 12와 같이 일반적으로 웹 서버에 요청하는 사용

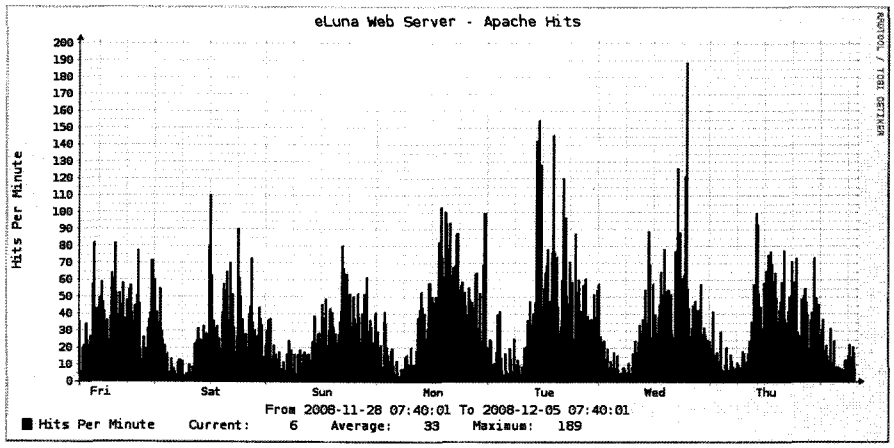


그림 12 웹 서버 요청량

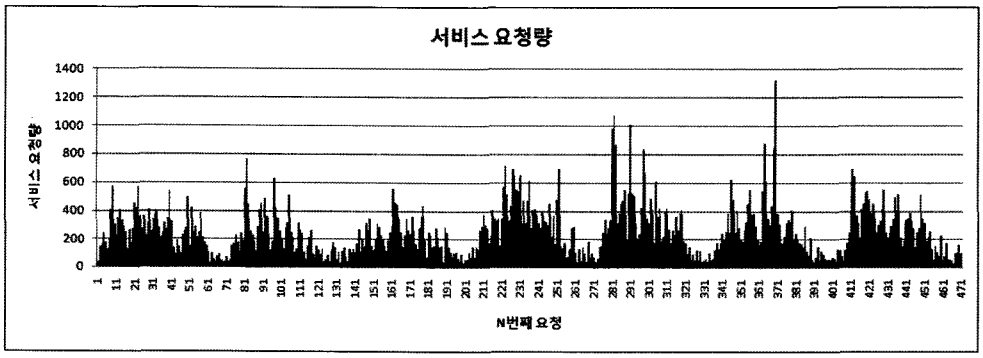


그림 13 사용자 서비스 요청량 그래프

자의 요청량[11]을 기반으로 성능을 측정한다.

그림 13의 사용자 요청량은 그림 12를 기반으로 본 실험에 사용된 서비스 요청량을 나타내며, 실험에 사용된 사용자의 요청량은 총 122,010개의 서비스의 요청을 470회에 걸쳐 하였다. 웹 서버로의 원하는 양의 서비스 작업을 요청하기 위해 웹 서버 벤치마크 툴인 httperf를 이용하였다. httperf[12]는 웹 서버의 성능 평가를 위해 다양한 HTTP 워크로드를 생성한다. 본 실험에서는 서버와 맺게 되는 세션의 총 수와 단위 시간(sec)당 연결 속도를 정해줌으로써 해당 서버의 처리 능력을 평가하였다. 성능 결과는 단위 시간당 서버의 응답 개수(reply/sec)로 나타난다. 따라서, 본 실험에서 평가하는 대상은 httperf에 의해 생성된 작업 요청이 웹 서버에서 모두 처리되고 난 후에 결과로 나타나는 응답 시간을 사용한다.

다음 절에서는 그림 13을 기반으로 서비스 처리를 요청함으로써, 각 기법에 대해 사용자 응답 시간을 기준으로 성능을 비교하고, 각 기법을 통해 웹 서버를 운영함으로써 사용되어지는 시스템 자원 사용율에 대해 성능을 비교한다. 보다 정확한 측정을 위해서 각 성능 평가 비교 대상들을 10회 반복하여 평균값으로 측정하였다.

5.2 실험 및 성능 분석

성능 평가의 기준으로는 응답 시간과 이전 주기에 예측한 값과 실제 사용된 쓰레드 수를 비교하여 예측의 정확성을 판단하였다. 또한, 동적으로 쓰레드의 수를 조절하는 것이 시스템 자원을 얼마나 효율적으로 사용하는지를 측정하기 위해 메모리 사용량을 비교 분석하였다.

5.2.1 사용자 응답 시간에 따른 결과 및 분석

본 논문에서 제안하는 기법과 지수 평균 기법의 동적 쓰레드 풀, 그리고 워터 마크 쓰레드 풀의 평균 응답 시간과의 차이를 계산하여 각 기법으로 웹 서버를 가동하였을 때 서비스를 요청한 시기에 응답 시간이 얼마나 최소화되었는지를 확인한다. 그림 14는 본 논문에서 제안한 동적 쓰레드 풀 기법을 통해 얻어진 사용자 응답 시간이 평균값에 비해 얼마나 최소화되었는지를 보여주고, 그림 15는 지수 평균 기법을 통한 동적 쓰레드 풀 구현 기법의 평균값에 따른 사용자 응답 시간을 보여준다. 그리고 그림 16은 워터 마크 쓰레드 풀 기법에 대한

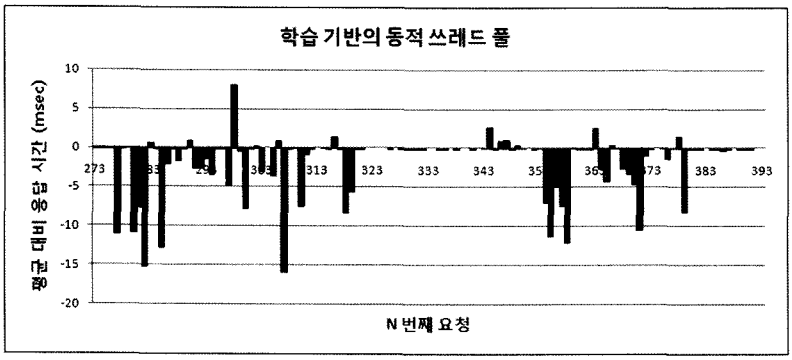


그림 14 최근접 이웃 알고리즘을 통한 응답시간

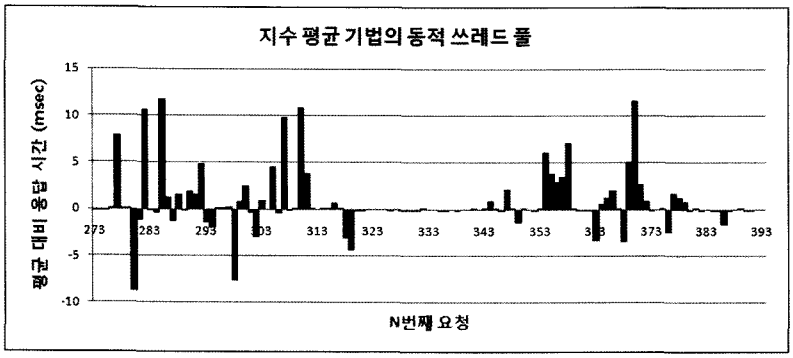


그림 15 지수 평균 기법을 통한 응답시간

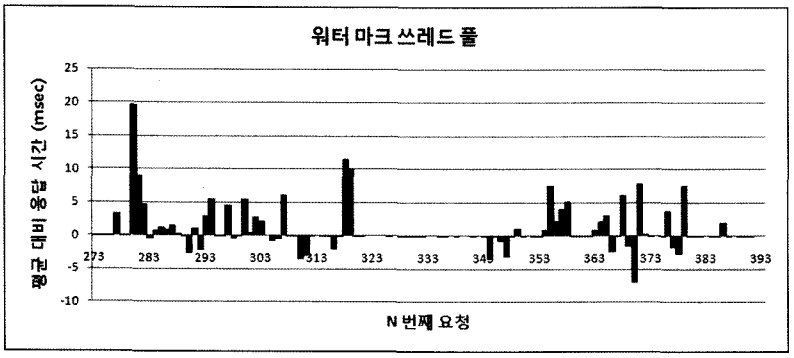


그림 16 워터 마크 쓰레드 풀 기법을 통한 응답시간

응답 시간을 나타내고, 그림 17은 쓰레드의 수를 8,192개로 고정시킨 워커 쓰레드 풀 기법의 사용자 응답 시간을 나타낸다.

본 논문에서 제안하는 기법인 학습 기반의 동적 쓰레드 풀 기법의 경우에는 서비스 요청량의 증감 추세가 없어도 경험을 기반으로 서비스 요청량에 따른 쓰레드를 결정함으로써 보다 정확한 수의 쓰레드 생성이 이루어졌다. 따라서 실제 서비스 요청시 추가적으로 생성하는 쓰레드의 양을 최소화함으로써 사용자에 대한 응답 시간을 줄일 수 있음을 알 수 있다.

하지만 이러한 기법을 적용시키지 않은 워터 마크 쓰레드 풀 기법의 경우에는 사용자의 요청이 이루어질 때마다 계속해서 쓰레드를 생성하고 삭제시킴으로써 이로 인한 과부하로 인해 응답 시간이 늦어짐을 알 수 있다. 그리고 쓰레드 풀의 크기를 최대로 고정하여 운영한 워커 쓰레드 풀 기법의 경우에는 항상 사용자의 요청보다 많은 쓰레드의 수를 유지를 하고 있기 때문에 다른 기법들에 비해 사용자 응답 시간이 가장 최소화될 수 있었다.

그림 18은 전체 서비스 요청량에 대한 평균 사용자

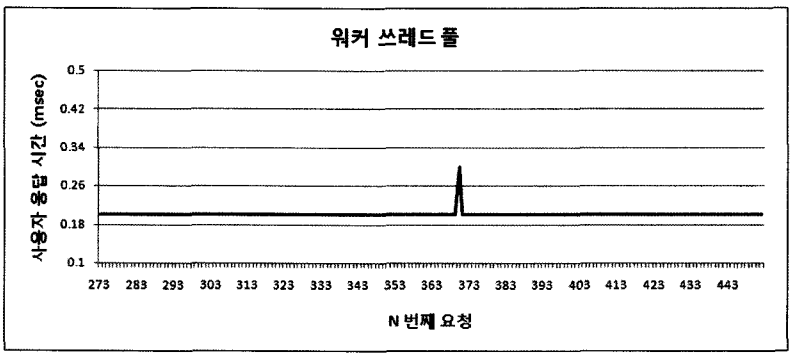


그림 17 워커 쓰레드 풀 기법의 사용자 응답시간

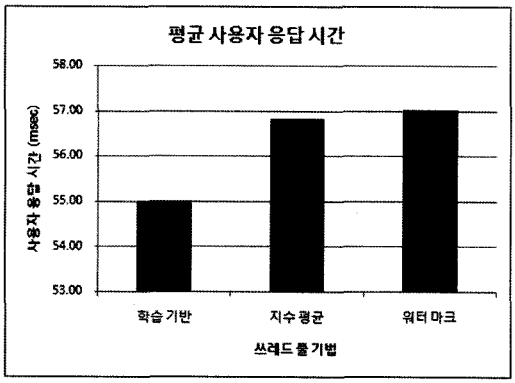


그림 18 평균 사용자 응답 시간

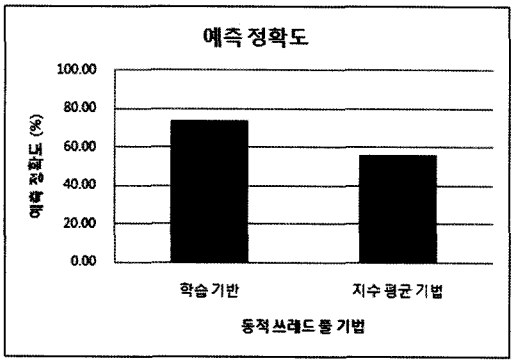


그림 19 예측 정확도

응답 시간을 나타낸 결과로 각각의 평균 사용자 응답 시간은 54.99msec, 56.82msec, 57.02msec, 0.2msec이다. 그리고 그림 19는 전체 서비스 요청에 대해 예측이 얼마나 정확하게 나타나고 있는지를 나타내는 그래프이다. 이는 이전 주기에 예측되어진 쓰레드의 수와 실제 사용되어진 쓰레드 수를 비교하여 예측의 정확성을 판단하였다.

학습 기반의 최근접 이웃 알고리즘을 사용한 동적 쓰레드 풀 기법은 사례 베이스에 서비스 요청량에 따라 실제로 사용되어진 쓰레드의 수가 저장되어 있기 때문에 그 값을 참조해서 보다 정확한 수의 필요한 쓰레드의 양을 결정할 수 있다. 그림 19에서 볼 수 있듯이 제안된 기법은 약 73.4%의 정확성을 보이고, 비교 대상인 지수 평균 기법의 동적 쓰레드 풀 기법은 약 55.8%의 정확성을 나타내었다.

위의 실험은 일반적인 사용자의 요청량을 기반으로 측정된 결과이다. 특정 구간에서 많은 요청량이 발생할 경우등 특수한 상황에서도 제안된 기법은 워터마크 기법보다 성능이 높다. 워터마크 기법의 경우, 높은 워터마크보다 많은 요청량이 발생하면 새로운 쓰레드를 생성함으로써 응답시간은 늘어나게 된다. 그러나 학습 기반의 동적 쓰레드 풀 기법을 사용할 경우, 쓰레드의 개수를 예측하고 쓰레드 미리 생성하기 때문에, 이로 인한 응답시간은 워터마크 기법에 비해 짧게 된다.

따라서, 본 논문에서 제안한 학습 기반의 동적 쓰레드 풀 기법이 지수 평균 기법을 사용한 동적 쓰레드 풀 기법에 비해 상대적으로 예측의 정확성이 높았기 때문에 불필요한 쓰레드를 생성하거나, 필요한 양보다 적은 쓰레드를 생성하여 추가적으로 생성해야 하는 과부하를 줄일 수 있어서 사용자 응답 시간을 좀 더 최소화 시킬 수 있었음을 알 수 있다.

5.2.2 효율적인 시스템 자원의 활용 결과 및 분석

그림 20은 각 기법으로 웹 서버를 가동하였을 때 사용되어진 메모리 사용율에 대한 그래프이다. 평균적으로 각 기법들은 36MB, 44MB, 16MB, 99MB의 메모리를 사용한다.

학습에 의해 결정되어지는 쓰레드의 양이 지수 평균 기법보다 예측이 더 정확하였기 때문에 불필요한 쓰레드를 생성하지 않게 되어 메모리 사용을 최소화함으로써 시스템 자원이 더 효율적으로 관리되어질 수 있음을 알 수 있다. 지수 평균 기법을 사용한 동적 쓰레드 풀 기법은 예측의 정확도가 많이 떨어짐으로써 실제 필요

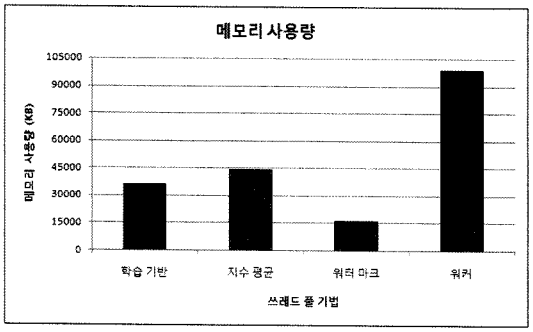


그림 20 평균 메모리 사용량

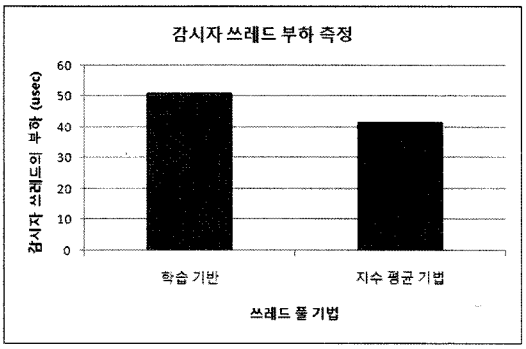


그림 21 감시자 쓰레드의 부하 측정

한 쓰레드의 수보다 적게 할당하거나 많이 할당하는 경우가 발생하였다. 실제로 본 실험에서 약 30%정도 불필요한 쓰레드의 생성으로 인해 메모리 사용률이 높은 것으로 나타났다.

워터 마크 쓰레드 풀 기법은 사용자의 요청량이 발생할 때마다 요청량만큼 쓰레드의 생성과 삭제를 반복하므로 불필요한 메모리 점유 현상이 발생하지 않아 가장 적은 양의 메모리를 사용한다. 그리고 쓰레드 풀의 크기를 고정적으로 사용한 워커 쓰레드 풀 기법은 쓰레드의 수를 최대로 생성하여 유지를 하기 때문에 응답 시간은 다른 기법에 비해 빠를지라도 한정된 자원으로 운영되어지는 시스템 환경에서 사용하지 않은 많은 메모리의 양을 불필요하게 고정적으로 점유하여 시스템 자원을 가장 비효율적으로 사용하고 있음을 알 수 있다.

이 실험에서 사용한 요청의 패턴은 그림 11과 같이, 일반적인 사용자의 요청량을 기반으로 측정되었고, 제안된 기법의 성능이 응답시간과 메모리 사용량 등이 다른 기법에 비해 향상되었다. 만약 요청이 지속적으로 발생하여 요청량이 증가하는 경우, 일반 요청량의 경우보다 더 안 좋은 결과를 낼 것이다. 워터마크의 경우, 높은 워터마크보다 요청량이 크면 더 많은 쓰레드를 생성함으로서 응답시간이 늘어나게 된다. 제안된 기법을 사용할 경우, 증가하는 요청량에 대해서 쓰레드의 수를 미리 예측하고 생성하기 때문에 응답시간이 워터마크 기법보다는 응답시간이 짧다.

5.2.3 감시자 쓰레드의 부하 측정

본 논문에서 제안한 학습 기반의 동적 쓰레드 풀 기법은 일정 간격으로 감시자 쓰레드를 통해 운영되는데, 감시자 쓰레드가 한번 수행할 때 걸리는 부하를 측정하고자 한다.

그림 21은 감사자 쓰레드의 부하를 측정한 결과이다. 실험을 통해 감시자 쓰레드가 한번 수행하는데 걸리는 시간은 본 논문에서 제안하는 기법은 평균 50.8×10^{-6} 초, 지수 평균 기법의 동적 쓰레드 풀 기법은 평균 41.3×10^{-6} 초가 소요됨을 확인하였다. 기존 기법에 비해 제안한 기법이 약 10×10^{-6} 초의 부하가 있기는 하였지만, 성능 평가 기준들이었던 사용자 응답시간이나 메모리 사용률에 있어서 제안하는 기법이 좀 더 효율적인 성능을 나타낸 것으로 보아, 감시자 쓰레드의 수행에 따른 부하가 제안하는 기법에 크게 영향을 미치지 않음을 볼 수 있다. 그리고 추가적인 쓰레드 생성을 하기 위해 자식 프로세스를 하나 생성하고 설정된 Threads-PerChild의 수만큼 쓰레드를 생성하는 시간은 평균 $1{,}904\times10^{-6}$ 초가 소요되었다. 이를 통해 감시자 쓰레드가 수행하는 시간이 추가적인 쓰레드를 생성하는 시간보다 훨씬 작은 시간임을 알 수 있었고, 필요한 쓰레드의 수를 학습하여 설정하는 방법이 서비스 요청이 올 때마다 쓰레드를 생성하는 방법보다 응답 시간을 훨씬 줄일 수 있음을 알 수 있다.

6. 결론 및 향후 과제

네트워크의 급속한 발전에 따라 사용자 수가 급속하게 늘어나게 되면서 사용자에 대한 응답 시간을 최소화하기 위한 방법이 많이 개발되어져 왔다. 그 중, 한정된 시스템 자원을 사용하는 임베디드 시스템에서는 다중 쓰레드 환경에서 효율적인 자원 관리를 통해 쓰레드를 미리 생성하여 사용자의 서비스 요청에 따라 즉시 처리가 가능한 다양한 쓰레드 풀 기법이 연구되어지고 있다.

본 논문에서는 불필요한 시스템 자원 점유 현상을 방지하고, 서비스 응답시간을 최소화 하기 위한 학습 기반의 쓰레드 풀 기법을 제안하였다. 이는 사용자의 요청량에 따른 쓰레드의 수를 학습함으로써 이전에 처리 되었던 경험을 기반으로 하여 필요한 쓰레드의 수를 결정할 수 있으며, 일정 주기의 증가나 감소하는 패턴을 가지지 않는 작업 요청량에도 대응할 수 있다. AR 기법을 이용해서 사용자 요청량을 예측하고, 최근접 이웃 알고리즘을 사용하여 가장 유사한 개체를 통해 필요한 쓰레드의

수를 설정함으로써 실제 사용자의 요청이 왔을 때 추가적인 쓰레드의 생성 시간을 감소시킴으로써 사용자에 대한 응답 시간을 최소화시킬 수 있음을 실험을 통해서 그 성능을 확인하였다. 또한, 사용자의 요청량에 맞게 쓰레드의 수를 관리함으로써 시스템 자원의 활용도도 높일 수 있었다.

본 논문에서 사용된 사용자 요청량은 일반적으로 웹 서버에 요청하는 서비스 양을 기준으로 하였으나, 사용자가 실제 상용 서버에 요청하는 요청량에 따라 시간적으로 오랜 시간 성능을 측정하거나 여러 서비스를 제공하는 일반 서버 환경에서 성능을 측정하여 제안하는 기법이 좀 더 효율적임을 보이는 방법에 대해 추후 연구가 더 필요하다.

참 고 문 헌

[1] A. Silberschatz, P. Galvin, G. Gagne, Operating System Principles, 7th edition, p.136, 2006.

[2] Yibei Ling, Tracy Mullen, and Xiaola Lin, Analysis of Optimal Thread Pool Size, *ACM SIGOPS Operating Systems Review*, vol.34, Issue 2, pp.42-55, February 14, 2000.

[3] D. Schmidt and S. Vinoski, "Object Interconnections: Comparing Alternative Programming Techniques for Multi-threaded CORBA Servers," *SIGS C++ Report magazine*, vol.8, no.4, April 1996.

[4] DongHyun Kang, Saeyoung Han, SeoHee Yoo, Sungyong Park, Prediction-based Dynamic Thread Pool Scheme for Efficient Resource Usage, *2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, pp.159-164, 2008.

[5] G. Coulouris, J. Dollimore, T. Kindberg, Distributed Systems Concepts and Design, 4th edition, p.235, 2005.

[6] A. Silberschatz, P. Galvin, G. Gagne, Operating System Principles, 7th edition, p.156, 2006.

[7] Apache 2.2.10, www.apache.org

[8] AutoRegressive, http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/ar/

[9] Jian Zhang and Renato J. Figueiredo, Adaptive Predictor Integratin for System Performance Prediction, *IPDPS 2007, IEEE*, pp.1-10, 2007. 3.

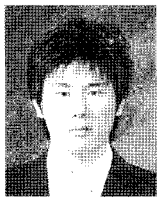
[10] 이창환, 정보이론을 이용한 K-최근접 이웃 알고리즘에서의 속성 가중치 계산, 정보과학회 논문지, 소프트웨어 및 응용 제 32권 제9호, pp.920-926, 2005. 9.

[11] 웹서버 요청량, http://graphs.eluna.org/

[12] httperf, http://www.hpl.hp.com/research/linux/httperf/

유 서 희

2009년 서강대학교 컴퓨터공학과 대학원 석사과정 졸업(공학석사). 2008년~현재 삼성경제연구소. 관심분야는 시스템 가상화, 임베디드 시스템



강 동 현

2008년 서강대학교 컴퓨터공학과 졸업(공학사). 2008년~현재 서강대학교 컴퓨터공학과 대학원 석사과정. 관심분야는 시스템 가상화, 임베디드 시스템

이 권 용

2007년 서강대학교 컴퓨터공학과 졸업(공학사). 2009년 서강대학교 컴퓨터공학과 대학원 석사과정 졸업(공학석사). 2008년~현재 서강대학교 컴퓨터공학과 대학원 박사과정. 관심분야는 시스템 가상화, 임베디드 시스템

박 성 용

1987년 서강대학교 컴퓨터학과(공학사) 1994년 미국 Syracuse University 대학원(공학석사). 1998년 미국 Syracuse University(공학박사). 1998년~1999년 미국 Bell Communication Research 연구원 1999년~2008년 서강대학교 컴퓨터학과 부교수. 2008년~현재 서강대학교 컴퓨터학과 정교수. 관심분야는 Autonomic Computing, Peer to Peer Computing, High Performance Cluster Computing and System