효율적인 자원 사용을 위한 예측기반 동적 쓰레드 풀 기법

(Prediction-based Dynamic Thread Pool Model for Efficient Resource Usage)

정 지 훈 † · 한 세 영 †† 박 성 용 †††
(Jihoon Jung) (Saeyoung Han) (Sungyong Park)

요 약 본 논문에서는 다중 쓰레드 기반의 서버 프로그래밍을 위한 예측 기반의 동적 쓰레드 풀 기법을 제안하였다. 기존의 쓰레드 풀(Thread Pool) 모델은 고정된 쓰레드 풀(Bounded Thread Pool)모델로, 서버에 최적화된 개수의 쓰레드를 유지하면서 다수의 요청에 대해 빠르게 응답할 수 있다는 장점이 있으나, 사용자의 접속이 적을 경우에도 고정된 시스템 자원을 점유하여야 하므로, 자원의 효율적인 사용에 문제가 있다. 이를 극복하기 위해 동적으로 쓰레드 풀의 크기를 변경하는 워터마크 쓰레드 풀 모델이 개발되었고, 본 논문에서는 이를 기반으로 지수평균을 사용하여 앞으로 필요한 쓰레드의 개수를 예측하고 쓰레드를 사전에 생성하여 더욱 효율적으로 자원을 활용하도록 하는 예측기반의 동적 쓰레드 풀 기법을 제안하였다. 제안한 기법은 사용자의 요청이 많을 경우 사용자 요청에 대한 응답시간을 빠르게 하고, 요청량이 적을 경우에는 불필요한 자원을 점유하지 않게 하여 기존의 워터마크 쓰레드 풀 모델에 비해 보다 성능이 좋고 자원을 효율적으로 활용함을 실험을 통해 확인하였다.

키워드 : 서버 설계, 쓰레드 풀, 지수 평균

Abstract A dynamic thread pool model is one of a multi-thread server programming model that handles many requests from users concurrently. In most cases, bounded thread pool model is used for server programming. Because it reduces a thread overhead by creating a pool of threads in advance, it can response more quickly to users' requests. But this model always occupies system resources when there are small amount of requests, which prevents other applications from using available resources. In this paper, for the utilization of occupied but unused resources, we proposed and implemented a prediction-based dynamic thread pool scheme using customized exponential average. From the experiments, we show that this scheme outperforms bounded thread pool model and uses small resources.

Key words : Server Design, Thread Pool, Exponential Average

1. 서 론

오늘날의 급속한 네트워크의 발전으로 서버 - 클라이언트 방식의 서비스가 일반적인데, 이를 구현하기 위해서는 서비스를 제공해 주기 위한 서버 프로그래밍과, 서버에 접속해서 필요한 데이타를 찾아 서비스를 받는 클라이언트 프로그래밍이 필요하다.

그 중 서버 프로그램의 경우 기본적으로 동시에 여러 사용자의 접속을 받아 각자에게 동시에 서비스를 제공하는 병행성(concurrency)이 구현되어야하는데, 이는 여러 개의 프로세스(process)를 생성하여 각 사용자들의 요청에 응답하는 형태로 프로그램들이 개발되어져 왔다. 그러나 프로세스 생성 및 관리의 과부하(overhead)로 인해 오늘날에는 프로세스가 아닌 다중의 쓰레드(thread)를 이용하여 서버 프로그래밍을 하게 된다.

그러나 쓰레드라 하더라도 사용자 요청을 받을 때마다 쓰레드를 생성해야 한다면 응답 시간이 길어질 것이고, 특히 다수의 사용자들이 짧은 시간이 소요되는 일을 요청하는 경우에는 더욱 응답시간에 큰 영향을 미치게

· 본 연구는 한국과학재단 목적기초연구(R01-2003-000-10627-0) 지원으로 수행되었음
† 비 회 원 : LG전자 정보통신사업본부
yuandi@dcclab.sogang.ac.kr
†† 비 회 원 : 서강대학교 컴퓨터학과
syhan@dcclab.sogang.ac.kr
††† 종신회원 : 서강대학교 컴퓨터학과
parksy@ccs.sogang.ac.kr
논문접수 : 2003년 3월 27일
심사완료 : 2004년 1월 2일

될 것이다. 따라서 사용자의 요청에 대한 응답시간을 적게 유지하면서도, 쓰레드 생성과 관리의 과부하를 줄이기 위해 다양한 방법들이 나오게 되었으며[1], 그 중에서도 쓰레드 풀(thread pool)[2]을 이용한 방법이 가장 대표적으로 사용되고 있다.

쓰레드 풀을 이용한 방법은 미리 지정된 개수의 쓰레드를 생성하여 하나의 풀(pool)로 구성하고 사용자의 요청이 있을 때 대기 중인 쓰레드가 처리하게 함으로써 응답시간의 지연 요인을 제거한다. 그러나 고정된 크기의 쓰레드 개수를 유지하기 때문에 실제 사용자의 요청이 쓰레드의 개수보다 적을 경우에도 시스템 자원을 점유하고 있어야 하므로 다른 프로그램들이 이 자원을 사용하지 못한다는 제약이 생긴다. 이는 특히 제한된 자원을 사용하는 임베디드 시스템(embedded system)에서나 자원의 효율적인 활용이 중요한 일반 범용 서버의 경우 치명적인 성능 저하 요인이 될 수 있다.

이런 여러 모델의 단점을 보완하여 동적으로 쓰레드 풀의 크기를 조정하는 워터마크 쓰레드 풀 모델(watermark thread pool model)[3]이 개발되었다. 이 모델에서는 쓰레드 풀에 높은 워터마크(high watermark)와 낮은 워터마크(low watermark) 값을 설정하고, 사용자의 요청이 적어 쓰레드 개수가 낮은 워터마크 이하일 경우에는 고정된 워커 쓰레드 풀 모델(woker thread pool model)[4]과 같이 동작하다가, 사용자 요청이 많아져 쓰레드 개수가 그 값을 넘어서면 요구기반 쓰레드 모델(thread per request model)[5]과 같이 필요한 쓰레드를 생성시켜 요청을 처리하게 된다. 단 이때 쓰레드의 개수는 높은 워터마크를 넘지 못하게 하여 쓰레싱(trashing)[6]으로 인한 시스템 자원 고갈을 막는다.

본 논문에서는 현재 접속하는 클라이언트의 수에 따라 쓰레드 개수를 동적으로 변화시키는 워터마크 쓰레드 풀 모델을 개선하여, 풀 내의 쓰레드 개수의 지수 평균(exponential average)[7,8]을 계산하여 필요한 쓰레드 수를 예측하고 미리 생성하는 예측 기반의 동적 쓰레드 풀 기법을 제안하였다. 풀 내 쓰레드 수의 지수평균을 계산하면서, 쓰레드 수가 증가하는 추세인 경우 쓰레드를 미리 생성하여, 사용자 요청이 왔을 때 쓰레드 생성으로 인한 응답 시간 지연을 방지하고, 또 한번 생성된 쓰레드는 서비스를 마친 후 바로 삭제하지 않고 일정시간 대기시켜 다른 사용자의 요청을 서비스 하는데 재사용될 수 있도록 한다. 서비스 요청이 일정시간 동안 없으면 비로소 그 쓰레드를 삭제하고 점유하였던 시스템의 자원을 다른 프로그램에서 사용할 수 있도록 돌려준다.

제안하는 기법을 구현하고 워터마크 쓰레드 풀 모델과의 성능을 비교하고자, 윈도우 운영체제 위에서 Java를 사용하였고, 사용자 요청에 대한 응답시간과, 메모리의 평균 사용량을 측정하여 두 모델을 비교, 분석하였다. 실험 결과 사용자의 요청이 많아 워터마크 쓰레드 풀이 요구기반 쓰레드처럼 동작하는 경우에는 제안하는 예측기반의 동적 쓰레드 풀의 응답시간은 훨씬 적은데 비해 메모리의 사용량은 약간 더 많았다. 그리고 사용자의 요청량이 적은 두번째 실험에서는, 워터마크 쓰레드 풀이 고정된 쓰레드 풀처럼 동작하므로 제안하는 예측기반의 동적 쓰레드 풀이 응답시간은 느리지만 메모리 사용량은 훨씬 적게 나타났다.

사용자의 요청량이 많아지면 스케줄링 해야 하는 쓰레드의 수가 많아져 응답시간 늘어나므로, 시스템의 자원을 좀 더 사용하더라도 응답시간을 조금이라도 줄이는 것이 바람직하고, 요청량이 적은 상황에서는 오히려 제한된 시스템의 자원을 사용하여 여러 서비스를 해야 하는 임베디드 시스템이나 일반 범용 서버의 경우, 사용자의 요청이 적을 때 최대한 시스템 자원 반환하여 다른 프로세스에서 활용하도록 하는 것이 바람직한 시스템의 설계 방향이 될 것이다. 따라서 본 논문에서 제안하는 예측기반의 동적 쓰레드 풀이 요구하는 시스템의 설계 방향에 좀더 부합함을 알 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 논문 배경으로서 기존의 서버 쓰레드 모델에 대한 대표적인 연구들을 소개하고 각각의 장단점 살펴보았고, 3장에서는 본 논문에서 제안하는 예측기반의 동적 쓰레드 풀 기법에 대해 자세히 서술하였다. 4장에서는 실험을 통해 타 모델과의 성능을 비교하고 결과를 분석한 후, 마지막으로 5장에서 결론 및 향후 과제에 대해 기술하였다.

2. 관련 연구

많은 다중 사용자의 요청을 동시에 처리해야 하는 서버 프로그래밍의 경우, 단위 시간당 처리량을 높이고, 사용자의 요청에 대한 응답시간을 줄이며, 자원 사용량을 줄이는 등 서버의 성능을 향상시키기 위해 다중 쓰레드(multi-threaded) 기반의 서버 모델들을 사용해 오고 있다[1]. 그 대표적인 구조로는 요구기반 쓰레드 모델(thread per request model)[5], 워커 쓰레드 풀 모델(worker thread pool model)[4], 그리고 워터마크 쓰레드 풀 모델(watermark thread pool model)[3]이 있다.

2.1 요구기반 쓰레드 모델(Thread per Request Model)

요구기반 쓰레드 모델(thread per request model)[5]은 사용자의 요청을 받아 요청을 처리할 워커 쓰레드를 생성하고 사용자의 요청을 넘겨주는 역할을 하는 입출력 쓰레드와 입출력 쓰레드에 의해 생성되어 사용자의 요청을 처리하고 소멸되는 워커 쓰레드로 구성된다. 이 모델의 경우 구조가 단순하여 구현이 용이하고 하나의 요청

을 처리하는데 비교적 많은 시간이 걸리는 데이타베이스 검색이나 파일 전송 등을 서비스하는 경우에 적합한 모델이라 할 수 있다. 그러나 사용자의 요청이 있을 때 새로운 쓰레드를 생성하여 해당 요청을 처리하고 쓰레드를 삭제해야 하므로 쓰레드 생성으로 인한 응답속도의 저하 및 계속되는 쓰레드 생성 및 삭제로 인한 과부하로 사용자 요청이 많은 경우에는 비효율적인 구조라고 할 수 있다. 더욱이 쓰레드 수의 제한이 없어 사용자 요청이 많아 쓰레드를 일정 수준 이상 생성하게 되면 CPU 시간이나 메모리 등과 같은 시스템의 자원을 소진하여 시스템 전체의 성능을 저하시킬 위험이 있다[6].

2.2 워커 쓰레드 풀 모델(Worker Thread Pool Model)

이런 제약을 가진 요구기반 쓰레드 풀 모델의 대안으로 워커 쓰레드 풀 모델(worker thread pool model)[4]이 개발되었다. 이 모델은 서버 시작 시 일정한 개수의 워커 쓰레드를 미리 생성하여 쓰레드 풀을 구성한다. 입출력 쓰레드에서 사용자의 요청을 받으면 쓰레드 풀에서 하나의 워커 쓰레드를 그 요청을 처리하도록 할당하고, 처리가 끝나면 그 워커 쓰레드를 다시 쓰레드 풀에서 대기하도록 한다. 이 모델은 워커 쓰레드의 생성 및 삭제에 따른 과부하를 없앰으로써 응답시간을 단축하고, 과도한 쓰레드 생성으로 인한 시스템 자원 고갈의 위험을 없앤다. 그러나 워커 쓰레드의 개수를 고정시키기 때문에 사용자의 요청에 비해 워커 쓰레드가 너무 적거나 많은 경우 유연하게 대처할 수 없다. 즉, 사용자의 요청이 적은 경우에도 일정 개수의 워커 쓰레드를 유지하므로 일정한 시스템 자원을 할당해야 하고, 사용자의 요청이 많은 경우 할당할 수 있는 시스템 자원이 남아 있더라도 사용자의 요청이 워커 쓰레드가 앞의 일을 마치기를 대기하고 있어야 하는 등 시스템의 자원을 효율적으로 활용할 수 없다.

2.3 워터마크 쓰레드 풀 모델(Watermark Thread Pool Model)

마지막으로 크기가 고정된 워커 쓰레드 풀 모델을 개선한 형태로 워터마크 쓰레드 풀 모델(watermark thread pool model)[3]이 있다. 이 모델에서는 쓰레드 풀의 크기를 사용자 요청량에 따라 동적으로 변하게 함으로써, 사용자 요청에 대한 응답시간은 적게 유지하면서 시스템의 자원을 효율적으로 활용하고자 개발된 모델이다. 이는 워커 쓰레드 풀에 낮은 워터마크와 높은 워터마크를 설정하고, 서비스 시작 시 낮은 워터마크 만큼의 워커 쓰레드를 미리 생성한다. 사용자의 요청이 많아 낮은 워터마크 만큼의 쓰레드가 다 사용되고 있는 경우, 높은 워터마크 이전까지 사용자의 요청을 처리할 워커 쓰레드를 생성한다. 워커 쓰레드의 개수가 높은 워터마크에 이르면 더 이상 쓰레드를 생성하지 않음으로써 서버에서의 시스템 자원 고갈을 방지한다. 즉, 현재 사용 중인 워커 쓰레드의 개수가 낮은 워터마크 이하일 경우는 고정된 워커 쓰레드 풀 모델과 같고, 낮은 워터마크 이상 높은 워터마크 이하일 경우에는 요구기반 쓰레드 모델과 같이 동작한다.

워터마크 쓰레드 풀 모델의 경우 고정된 워커 쓰레드 풀 모델과 요구기반 쓰레드 모델의 단점들을 보완하여 나온 모델이긴 하지만, 요청량이 많을 경우에 사용자 요청시 워커 쓰레드 생성 과부하에 의한 응답시간 지연과 사용자 요청량이 적을 때 불필요하게 시스템 자원을 할당하고 있는 부분은 여전히 존재한다. 이는 적은 시스템 자원을 활용하여 서비스를 함으로써 자원의 효율적인 활용이 중요한 임베디드 시스템에서나 여러 종류의 서비스를 제공하는 범용 서버에서는 부적합한 모델이 될 것이다. 따라서 필요한 워커 쓰레드의 개수를 예측하고 생성해 놓음으로써 사용자의 요청이 있을 때 쓰레드 생성으로 인한 응답시간의 지연을 개선하고, 사용자의 요청이 없을 때 쓰레드를 삭제시키고 그 쓰레드의 자원을 다른 프로그램에서 사용할 수 있도록 하는 더 효율적인 구조에 대한 연구가 필요하다.

3. 예측기반의 동적 쓰레드 풀 기법

3.1 제안 기법의 구조

본 장에서는 사용자 요청량이 많은 경우 효과적으로 서버의 응답시간을 개선하고, 요청량이 적은 경우 효율적으로 시스템의 자원을 활용할 수 있도록 하는 예측기반의 동적 쓰레드 풀 기법을 제안하고자 한다.

본 제안 기법에서는 프로그램 시작 시 쓰레드 풀을 생성하고 사용자의 요청량에 따라 풀의 크기가 동적으로 변하도록 설계하였다. 즉, 입출력 쓰레드에서 사용자의 요청을 받으면, 먼저 쓰레드 풀에 대기중인 워커 쓰레드가 있는지 확인하여 사용자의 요청을 처리할 워커 쓰레드를 할당하고, 만일 대기 중인 워커 쓰레드가 없다면 새로운 쓰레드를 생성하여 사용자의 요청을 처리한다. 여기까지의 사용자 요청을 처리하는 방법은 요구기반 쓰레드 모델과 유사하다. 본 기법에서는 여기에 쓰레드 풀 감시자를 두어, 주기적으로 쓰레드 활성화나 생성 및 삭제 시에 기록해 놓은 현재 쓰레드 개수(t_n)를 읽어, 다음 주기에 필요한 쓰레드 수인 현재의 예측값(Γ_{n+1})을 지수평균[8]을 이용하여 계산하고, 이 값이 현재 쓰레드 개수 보다 많을 경우 그 만큼의 워커 쓰레드를 미리 생성하여 풀에서 대기시킨다. 또한 각 워커 쓰레드에는 최대 대기시간 초과 타이머(idle timeout timer)를 구현하고 일정시간 동안 풀에서 대기하면 삭제되도록 함으로써, 사용자의 요청이 적을 때 쓰레드 풀의 크기를 줄여나간다.

그림 1은 예측기반의 동적 쓰레드 풀 기법의 구조를

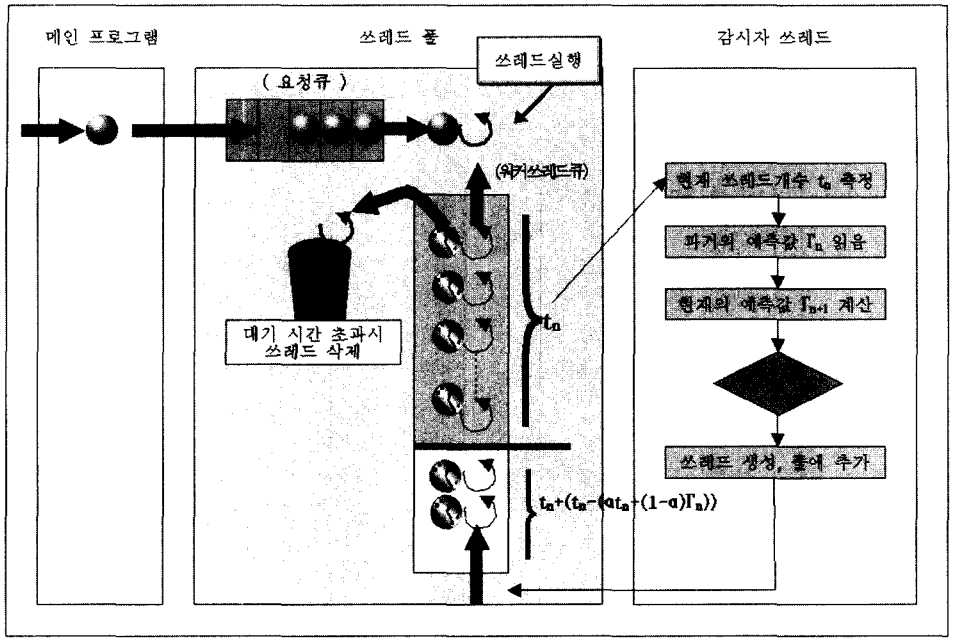

그림 1 예측기반 동적 쓰레드 풀 기법 개요

보여주고 있다. 예측기반의 동적 쓰레드 풀 기법은 크게 메인 프로그램, 쓰레드 풀, 워커 쓰레드, 그리고 감시자 쓰레드로 구성되어 있다.

메인 프로그램에서는 서비스 시작 시 쓰레드 풀을 생성하고, 서버의 서비스가 종료될 때 까지 사용자의 요청이 들어오기를 기다려 그 요청 정보를 요청큐에 추가하고, 쓰레드 풀에게 알리는 역할을 수행한다. 쓰레드 풀은 워커 쓰레드 큐를 관리하며, 사용자 요청이 왔다는 정보를 받으면 쓰레드 큐에서 대기 중인 워커 쓰레드를 하나 선택하여 그 요청을 처리하도록 활성화시키고, 대기 중인 쓰레드가 없을 경우 쓰레드를 생성하여 요청을 처리하도록 한다. 요청의 처리를 마친 워커 쓰레드는 풀에서 대기 상태로 다음 요청을 기다리는데, 대기 상태로 일정시간이 지나면 삭제된다. 이때 워커 쓰레드의 생성과 삭제 시에 현재 동작하고 있는 쓰레드의 개수를 측정하여 기록해 놓으면, 감시자가 주기적으로 이 기록된 정보를 기반으로 지수평균을 계산하여 다음 주기에 필요한 워커 쓰레드 개수를 예측하고, 쓰레드를 미리 생성하여 워커 쓰레드 풀의 큐에 추가한다.

각 구성 프로그램의 구조 및 구체적인 알고리듬은 다음과 같다.

3.1.1 메인 프로그램

메인 프로그램의 구조는 그림 2와 같다. 즉, 메인 프로그램이 실행되면 먼저 쓰레드 풀을 생성하고, 소켓 리스너 (Socket Listener)를 만들어서 사용자의 요청을 기다린다. 소켓 리스너에 새로운 사용자의 접속이 등록되면, 사용자의 소켓정보를 쓰레드 풀에 있는 요청 큐로 넘겨주고 쓰레드 풀에게 사용자 요청이 요청큐에 추가되었음을 알린 후, 다시 사용자 요청을 대기한다. 이처럼 메인 프로그램은 단순히 사용자의 접속을 받아들여

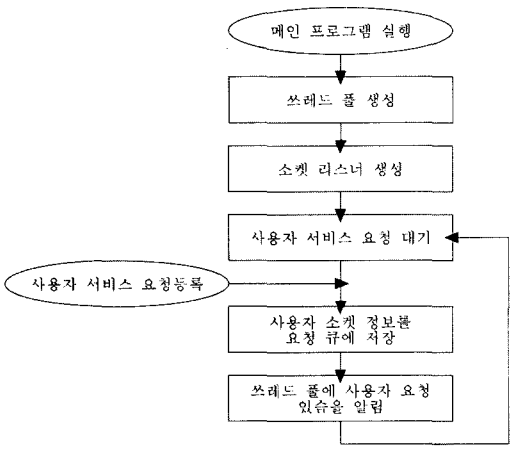

그림 2 메인 프로그램의 구조

생성된 풀에 사용자의 요청을 계속해서 추가하는 일을 하며, 필요한 경우에는 쓰레드 풀의 상황을 확인하는 역할도 하게 된다.

3.1.2 쓰레드 풀

쓰레드 풀은 처음 생성되면, 사용자의 소켓 정보들을 저장하기 위한 요청 큐와 쓰레드들을 관리하기 위한 워커 쓰레드 큐를 생성하고, 사용자 요청이 들어오기를 기다린다. 메인 프로그램으로부터 사용자의 요청이 들어왔다는 메시지를 받게 되면 먼저 요청 큐에 메인 프로그램으로부터 받은 사용자 요청 소켓정보를 추가한 후, 쓰레드 큐를 검색하여 대기 중인 워커 쓰레드를 깨운다. 대기중인 쓰레드가 없을 경우에는 새로운 쓰레드를 생성하고, 신호를 통해 쓰레드로 하여금 요청 큐에서 정보를 읽어 해당 요청을 처리하도록 한다. 쓰레드 풀 프로그램의 구조는 그림 3과 같다.

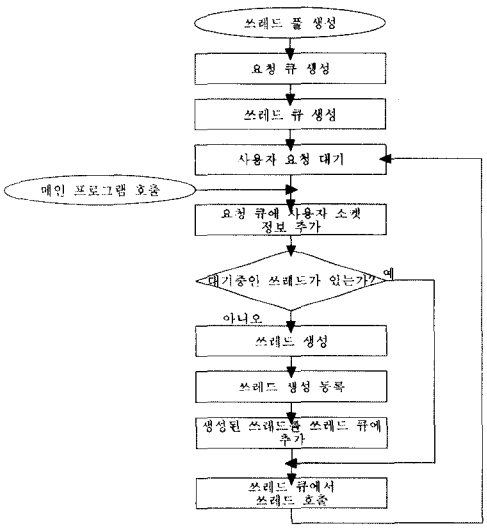

그림 3 쓰레드 풀 프로그램 구조

3.1.3 워커 쓰레드

각 워커 쓰레드의 경우 **대기상태와 활성상태**를 가지는데, 대기상태일 경우에는 일정시간이 지나면 자동으로 워커 쓰레드 큐에서 자신을 삭제하고, 현재의 워커 쓰레드의 개수를 측정하여 그 값을 기록한다. 대기상태에서 쓰레드 풀에 의해 깨워져 사용자의 요청을 처리하는 활성상태가 되면, 먼저 현재 동작중인 워커 쓰레드의 개수를 측정하여 기록한 후 사용자의 요청을 처리한다. 쓰레드가 요청을 실행하는 순서는 그림 4와 같고, 쓰레드를 삭제하는 알고리듬은 그림 5와 같다.

3.1.4 감시자 쓰레드

감시자 쓰레드의 경우 주기적으로 현재 기록된 워커 쓰레드의 개수를 읽어 다음에 필요한 쓰레드 예측값을

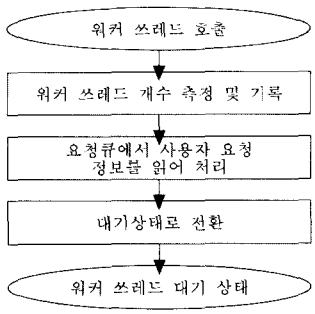

그림 4 워커 쓰레드 프로그램

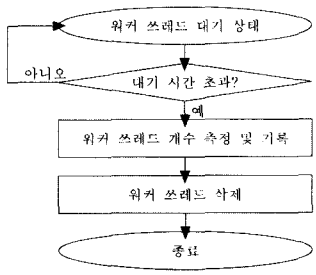

그림 5 워커 쓰레드 삭제 구조

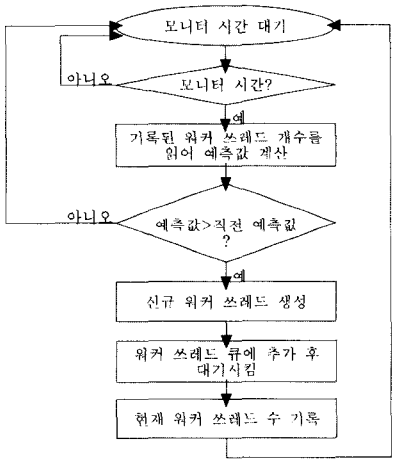

그림 6 감시자 쓰레드 구조

지수평균을 이용하여 계산하고, 바로 전 주기에서의 예측값과 비교하여 더 많을 경우 새로운 쓰레드를 미리 생성하고 워커 쓰레드 큐에 추가한다. 그림 6에서 상세 구조를 나타내었다.

3.2 쓰레드 변화량 측정 및 예측 기법

사용자의 요청량에 따라서 쓰레드의 개수를 동적으로 조절하면서, 사용자 요청량이 많은 경우 효과적으로 서버의 응답시간을 개선하고, 요청량이 적은 경우 효율적으로 시스템의 자원을 활용할 수 있도록 하기 위해서는

다음과 같은 사항들을 고려하여야 한다.

첫째, 요청량이 증가하는 추세일 경우에는 증가량의 정도를 파악해서 미리 쓰레드를 생성시킬 수 있어야 한다. 즉, 쓰레드 양이 급격히 증가하는 추세일 경우에는 그에 비례해서 많은 쓰레드를 미리 생성해야 한다.

둘째, 요청량이 감소하는 추세일 경우에는 쓰레드의 수를 급격하게 줄이지 않도록 해야 한다. 요청량이 줄어들 때 이에 맞추어 쓰레드의 수를 급격하게 감소시킨다면, 쓰레드 량이 다시 증가하는 경우 불필요한 쓰레드의 삭제 및 생성이 발생하게 된다. 이러한 현상을 막기 위해서는 쓰레드를 바로 삭제하지 않고 일정시간 유지되도록 해야 한다.

마지막으로, 쓰레드 증감의 정도는 과거의 값들이 반영 되어야 한다. 과거의 변화 정도를 반영하지 않고 단순히 현재의 변화 정도만을 가지고 다음 값을 예측한다면 일시적으로 값이 증가하거나 감소하는 경우에 불필요하게 쓰레드를 생성하거나 삭제하는 일이 빈번하게 발생하기 때문이다.

3.2.1 워커 쓰레드의 개수 측정 기법

현재의 워커 쓰레드 개수를 측정하기 위해서는 일정한 시간 간격마다 검사를 하는 방법과, 워커 쓰레드의 생성과 삭제시기에 검사를 하는 방법이 있다

그림 7은 워커 쓰레드의 생성과 종료 시 개수를 측정한 그래프인데, 이 경우 정확하게 워커 쓰레드의 생성을 반영하므로, 다음의 쓰레드 필요량을 보다 정확하게 예측할 수 있다. 또한 워커 쓰레드의 변화량이 적은 경우 주기적으로 쓰레드를 양을 검사하는 방법보다 과부하가 적다.

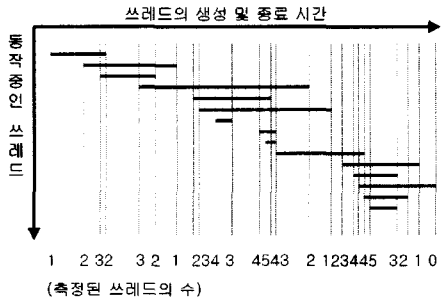

그림 7 쓰레드의 생성시기와 종료시기에 동작하는 쓰레드의 수를 검사하는 경우

3.2.2 워커 쓰레드 변화량 예측 기법

일정한 패턴이 없이 불규칙적인 증감이 많은 데이타량을 예측해야 하는 경우, 특정 값을 선택할 수 없고 평균값을 사용하게 된다. 이때 과거의 데이타들과 현재의 데이타를 일정 비율로 고려하는 지수 평균 기법을 사용하게 된다.

지수평균을 사용한 예로는 먼저 TCP 프로토콜의 재전송 기능을 들 수 있다. TCP 프로토콜에서 TCP 세그먼트를 전송한 후 일정시간 동안 응답이 없으면, 그 세그먼트를 재전송 하도록 시간초과(timeout) 값을 결정해야 하는데, 이는 그 TCP 연결(connection)의 왕복시간(round trip time)보다 너무 크거나 작지 않도록 조정되어야 한다. 그러나 라우터에서의 혼잡이나 시스템의 부하에 의해 왕복시간은 각 세그먼트마다 달라지므로, 이때 지수평균을 사용하여 다음 세그먼트의 왕복시간을 예측하게 된다[7].

또 다른 지수평균의 예로 CPU 스케줄링 방법 중에서 Shortest-Job-First Scheduling을 들 수 있다. 이는, 현재 대기 중인 프로세스 중에서 가장 빨리 끝날 프로세스를 선택하여 먼저 수행시킴으로써 전체 프로세스들의 평균 대기시간을 최소화하는 알고리듬인데, 프로세스가 끝나는 시간은 미래의 일로 정확히 알 수 없으므로, 과거 CPU 점유시간의 지수평균을 그 프로세스의 CPU 점유시간으로 하고, 가장 먼저 끝날 것으로 예상되는 프로세스를 선택하여 먼저 실행 시킨다. 이때 $\alpha = 0.5$인 지수 평균을 사용한다[8].

일반적으로 지수 평균을 구하는 방법은 다음과 같다.

$$\Gamma_{n+1} = \alpha t_n + (1-\alpha)\Gamma_n \qquad (0 \le \alpha \le 1)$$

여기서 t_n은 현재 시점에서의 측정값이고, Γ_n은 현재 시점에 대한 과거의 예측값, 그리고 Γ_{n+1}는 다음 시점에 대한 현재의 예측값을 나타낸다. α는 현재의 측정값과 과거의 측정값들을 어느 정도 비율로 반영하여 다음 시점의 예측값을 계산할 것인지를 결정하는 값이다.

α가 0인 경우 $\Gamma_{n+1} = \Gamma_n$이 되어 최근의 측정값이 전혀 반영되지 않으며, α가 1인 경우에는 $\Gamma_{n+1} = t_n$가 되어 최근의 측정값이 바로 예측값이 되게 된다. 일반적으로 쓰레드 개수 측정에 사용될 경우 $\alpha = 0.5$를 사용하는데, 이는 과거와 현재에 측정된 쓰레드 개수에 똑같은 비중을 두어 미래의 쓰레드 개수를 예측하게 한다. 본 논문에서도 $\alpha = 0.5$를 사용하여 일시적인 쓰레드의 증가나 감소의 영향을 적게 받게 하였다. 위의 식에서 Γ_n을 이전의 측정값들로 치환하면 아래와 같은 식이 나오게 된다.

$$\Gamma_{n+1} = \alpha t_n + \alpha(1-\alpha)t_{n-1} + \alpha(1-\alpha)^2 t_{n-2} + \cdots + \alpha(1-\alpha)^i t_{n-i} + \cdots + (1-\alpha)^{n+1}\Gamma_0$$

이 식에서 바로 직전에 측정한 값인 t_{n-1}은 $\alpha(1-\alpha)$ 만큼 반영되고, i번 전에 측정한 값인 t_{n-i}의 경우는 $\alpha(1-\alpha)^i$ 만큼이 예측값에 반영이 되는 것을 볼 수 있다. 그런데 $(1-\alpha)$은 1보다 작은 값이므로 현재와 가까운 측정값 일수록 많은 비중으로 반영되고, 더 옛날에 측정한 값일수록 적은 비중으로 예측값에 반영됨을 알 수 있다.

그림 8은 지수평균 기법으로 예측되는 값을 원래의

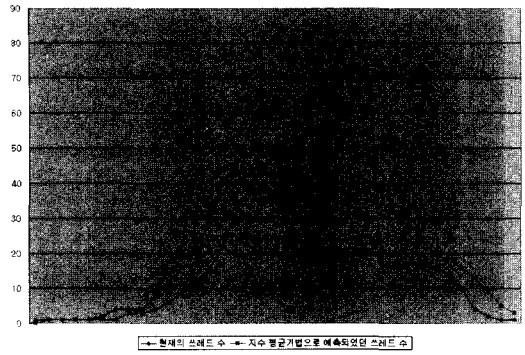

그림 8 지수 평균을 적용한 쓰레드의 변화량

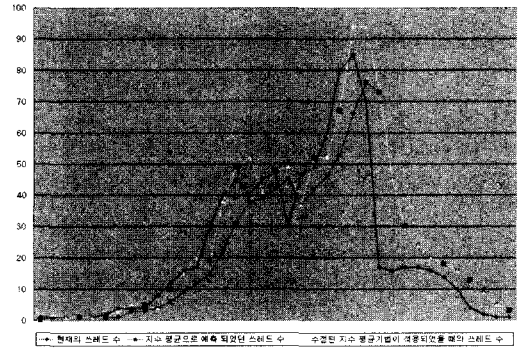

그림 9 수정된 지수 평균 기법을 적용한 쓰레드의 변화량

값과 같이 그래프로 나타낸 것인데, 이를 현재의 쓰레드 개수를 기반으로 다음의 쓰레드의 개수를 예측하는 상황으로 적용해 볼 수 있다. 지수평균의 특성상 현재의 값 뿐 아니라 과거의 측정값도 반영하기 때문에 일시적으로 급격히 증가하거나 감소하는 경우 이에 따라 예측값이 급격히 변하지는 않는다.

따라서 일시적으로 요청량이 증가하거나 감소하는 경우에도 일정한 수준으로 쓰레드의 개수를 유지함으로써 불필요한 쓰레드의 생성 및 삭제를 줄일 수 있도록 지수평균 기법을 이용할 수 있다.

그러나 이 그래프에서 볼 수 있듯이 쓰레드가 지속적으로 증가하는 구간에서는 다음 시점에서 현재보다 많은 수의 쓰레드를 필요로 하지만, 그 예측값은 현재의 측정값보다 적을 수밖에 없으므로, 쓰레드가 증가하는 추세일 경우 이 방법을 그대로 사용할 수는 없다. 따라서 본 논문에서는 이러한 문제점을 보완하여, 현재 지수평균에 의한 예측값인 Γ_{n+1}에서 전 단계 예측값인 Γ_n를 뺀 값으로 쓰레드의 증감 추이를 판별한 후 다음과 같이 쓰레드의 수를 예측하도록 원래의 지수 평균 식을 수정하였다.

1) $\Gamma'_{n+1} = t_n + (t_n - \Gamma_{n+1})$, $\Gamma_{n+1} > \Gamma_n$ 일 때,

2) $\Gamma'_{n+1} = \Gamma_{n+1}$, $\Gamma_{n+1} \leq \Gamma_n$ 일 때.

1)에서 $\Gamma_{n+1} > \Gamma_n$인 경우란, 현 시점에서의 지수평균 예측값 Γ_{n+1}이 직전 예측값인 Γ_n 보다 크다는 의미로, 쓰레드의 수가 증가하고 있다는 뜻이다. 이 때 지수평균 예측값 Γ_{n+1}은 현재 쓰레드 수 t_n 보다 작으므로, 즉,

$\Gamma_{n+1} = \alpha t_n + (1-\alpha)\Gamma_n > \Gamma_n \rightarrow \alpha(t_n - \Gamma_n) > 0 \rightarrow t_n - \Gamma_n > 0$

이므로,

$t_n - \Gamma_{n+1} = t_n - (\alpha t_n + (1-\alpha)\Gamma_n) = (1-\alpha)(t_n - \Gamma_n) > 0 \rightarrow t_n - \Gamma_{n+1} > 0$

이다. 따라서 현재 쓰레드 수에 지수평균 예측값과 현재 쓰레드 수의 차이 $(t_n - \Gamma_{n+1})$를 더하여 다음 예측값 Γ'_{n+1}을 결정함으로써, 단순한 지수평균의 예측값인 Γ_{n+1}보다 더 많은 수의 쓰레드를 미리 생성할 수 있도록 하였다.

이를 수식으로 확인하면 다음과 같다.

$\Gamma'_{n+1} - \Gamma_{n+1} = t_n + (t_n - \Gamma_{n+1}) - \Gamma_{n+1} = 2 * (t_n - \Gamma_{n+1}) > 0.$

2)에서는 쓰레드가 증가하고 있지 않은 경우 일반적인 지수 평균에 의한 예측값을 그대로 사용 하도록 하였다. 따라서 쓰레드 량이 감소하고 있더라도 쓰레드 량을 여유있게 남겨둠으로써 일시적인 쓰레드 증가에 효과적으로 대처할 수 있게 하였다.

그림 9는 변경된 수식으로 쓰레드 예측값을 결정하는 것을 앞의 그림 8 그래프에 추가하여 같이 나타내었는데, 지수평균 방법보다 효과적으로 미리 필요한 쓰레드를 생성할 수 있음을 볼 수 있다.

3.2.3 쓰레드 삭제 기법

쓰레드가 어떤 경로를 통해서든 생성되면, 쓰레드 큐에서 사용자의 요청을 기다리는 대기상태에 있거나 사용자의 요청을 처리하는 활성상태에 놓이게 된다. 활성상태에 있더라도 요청의 처리가 끝나면 대기상태로 돌아가 다음 사용자의 요청을 기다리는데, 사용자의 요청량이 적어 다음 요청이 일정 시간동안 오지 않으면, 즉, 쓰레드가 대기상태에서 일정 시간 이상 있게 되면 스스로를 삭제하게 된다. 이런 구조를 통해 불필요하게 쓰레드가 점유하고 있는 시스템의 자원을 다른 프로세스에서 활용할 수 있도록 한다.

4. 실험 및 성능 분석

본 장에서는 동적인 쓰레드 풀을 갖는 워터마크 쓰레드 풀과 본 논문에서 제안하는 예측 기반의 동적 쓰레드 풀의 성능을 측정하고 비교, 분석 하였다.

워터마크 쓰레드 풀 모델의 경우 기존의 고정된 쓰레드를 갖는 모델에서 사용자의 요청량이 설정된 수보다 너무 많거나 적은 경우에 시스템의 자원을 유연하게 활용할 수 없는 단점을 보완하여 개발된 모델로, 사용자 요청량에 따라 쓰레드의 크기를 동적으로 조정함으로써 사용자의 요청에 대한 응답시간을 적게 유지하면서 시

스템의 자원을 효율적으로 활용하도록 하는 모델이다.

워터마크 쓰레드 풀을 구현하기 위해서는 쓰레드 풀에 높은 워터마크 값과 낮은 워터마크 값을 설정해야 한다. 높은 워터마크 값은 그 시스템에서 쓰레싱 없이 운영할 수 있는 최대의 쓰레드 수로 설정하고, 사용자의 요청량이 증가하여 풀 내의 쓰레드 개수가 증가하더라도 이 값 이상으로는 늘리지 않도록 하여 시스템 자원의 고갈을 방지한다. 사용자 요청량이 적은 경우 워터마크 쓰레드 풀은 고정된 쓰레드를 갖는 풀로서 동작하다가, 사용자의 요청이 많아져 낮은 워터마크 값을 넘어서게 되면 들어오는 요청에 따라 새로운 쓰레드를 생성하고 요청을 처리하게 되어 마치 요구기반 쓰레드 풀과 같이 동작하게 된다.

따라서 본 장에서는 사용자의 요청량이 충분히 많아 워커 쓰레드 수가 낮은 워터마크 값보다 많은 워터마크 쓰레드 풀이 요구기반 쓰레드 모델과 같이 동작하는 경우와, 사용자의 요청이 충분히 적어 고정 쓰레드 풀처럼 동작하는 경우, 두가지 상황에서 본 논문에서 제안하는 예측기반의 동적 쓰레드 풀 기법과 성능을 비교하기 위하여, 두가지 상황에 대한 실험을 실시하였다. 그리고 성능평가는 사용자 요청에 대한 응답시간과 평균 메모리 사용량을 측정하여 비교하고 분석하였다.

4.1 실험 개요

본 실험을 위한 장비는 제안된 예측기반의 동적 쓰레드 풀 기법을 구현한 서버 시스템, 서버에 서비스 요청을 보내고 응답 시간을 측정하기 위한 사용자 시스템, 그리고 100Mbps의 스위치드 패스트이더넷(Switched FastEthernet) 네트워크로 구성 되어 있다. 서버 시스템은 펜티엄III 800MHz CPU와 256MB의 메모리, 사용자 시스템은 펜티엄III 800MHz SMP CPU와 512MB 메모리를 장착하였다. 본 실험을 위하여 사용자 시스템에서 단위 시간당 사용자의 수를 포아손(Poisson) 분포를 이용하여 결정한 후 산출된 만큼의 쓰레드들이 동시에 서버 접속을 시도하도록 하며, 서버에 접속한 쓰레드들은 100 KByte 크기의 패킷을 주고 받도록 한다.

그림 10은 실험에 사용된 부하량을 나타내는데, 포아손(Poisson) 분포를 따르는 데이타 집합을 생성하기 위하여 GTSS[9]에서 만들어진 수학 라이브러리를 이용하였다.

먼저 서버 시스템에서는 쓰레드 개수가 60개를 넘어가면 쓰래싱 현상으로 인한 과부하가 생김을 알 수 있었으므로, 이를 방지하기 위해서 최대 쓰레드의 개수를 50개로 설정하고, 부하량이 많아 요청량이 최대 풀 크기인 50개에 근접하는 경우부터 부하량이 적어 요청량이 0에 근접하는 경우까지 λ 값을 변경해가면서 그래프를 산출하고, 결과 값을 분석한 후, λ = 35와 λ = 10 인 경우를 추출하여 사용하였다.

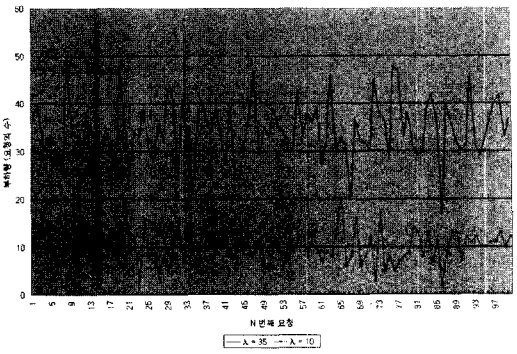

그림 10 부하량 그래프

4.2 요청량이 많은 경우의 실험 결과 및 분석

본 실험에서는 사용자 시스템에서 λ = 35로 생성된 데이타들을 두 서버 모델에 동일하게 적용시키기 위해 배열에 저장한 후, 0.1초 간격으로 서비스 요청을 시행하였다. 쓰레드의 특성상 서버에서 각 요청에 대한 실행 시작시간이 일정하지 않으므로 보다 정확한 측정을 위하여 위의 실험을 50회 반복하고 응답 시간과 메모리 사용량의 평균을 산출하였다. 감시자는 0.1초 마다 동작하도록 설정하였고, 쓰레드의 삭제 전 최대 대기시간(idle timeout)은 0.3초로 설정하였다.

그림 11과 그림 12는 실험 결과인 두 모델의 평균 응답시간과 평균 메모리 사용량을 그래프로 나타내었다. 그림 11의 그래프에서 본 논문에서 제안하는 예측 기반의 동적 쓰레드 풀이 워터마크 쓰레드 풀에 비해 응답시간이 월등히 좋아졌음을 알 수 있다. 이는 두 모델 다 사용자의 요구에 맞춰서 쓰레드를 생성하긴 하지만, 예측기반의 동적 쓰레드 풀은 사용자의 요청이 증가하여 쓰레드의 수가 증가하는 경우 미리 쓰레드를 생성시키고, 요청량의 감소로 인해서 쓰레드가 감소하는 경우에도 급격하게 쓰레드를 줄이지 않고 일정한 수준의 쓰레드를 유지하여 요청을 받는 시점에서 쓰레드 생성해야 하는 상황을 최소화시킨다. 따라서 높은 부하에서 사용자의 요청이 올 때마다 쓰레드를 생성해야하는 워터마크 쓰레드 풀 모델에 비해 월등히 좋은 응답시간을 나타내는 것이다.

그림 12는 두 모델이 동작하는데 사용된 메모리의 양을 100msec 단위마다 측정하여 평균을 낸 결과를 나타낸 그래프인데, 자바의 특성상 강제로 가비지 컬렉션을 해주지 않으면 메모리가 계속해서 증가하기 때문에 보다 정확한 측정을 위해서 강제로 가비지 컬렉션을 하도록 하였다.

실험 결과 예측기반 동적 쓰레드 풀이 워터마크 쓰레

드 풀에 비해 메모리 사용량이 더 많았다. 이는 예측기반 동적 쓰레드 풀의 경우 사전 예측을 통해 실제보다 많은 쓰레드를 생성하는 경우가 많고, 사용자의 요청을 처리한 후에도 바로 쓰레드를 삭제하지 않고 일정시간 대기 상태를 유지하기 때문이다. 그러나 사용자의 요청이 많은 경우에는 대기상태의 쓰레드가 거의 없을 것이므로, 두 모델 간의 메모리 사용량의 차이는 많지 않다.

결과적으로 사용자의 요청량이 많은 상황에서는 예측기반의 동적 쓰레드 풀이 워터마크 쓰레드 풀에 비해 메모리를 조금 더 많이 사용하면서 응답시간은 현저하게 줄일 수 있음을 알 수 있었다. 사용자의 요청량이 많아지면 스케줄링 해야 하는 쓰레드의 수도 많아지므로 응답시간 자체가 늘어난다. 이런 상황에서는 시스템의 자원을 좀 더 사용하더라도 응답시간을 조금이라도 줄이도록 하는 것이 바람직한 시스템의 설계 방향이라고 할 수 있다. 따라서 제안하는 예측기반의 동적 쓰레드 풀 기법이 사용자의 요청량이 많은 상황에서는 워터마크 쓰레드 풀 모델 보다 적합하다고 할 수 있다.

4.3 요청량이 적은 경우의 실험 결과 및 분석

본 실험에서는 λ = 10인 데이타를 사용하여 5초 간격으로 사용자 시스템이 서버에 서비스를 요청하도록 하였고, 역시 50회의 실험을 반복하여 응답시간과 메모리 사용량의 평균을 산출하였다. 앞의 실험에서와 마찬가지로, 감시자는 0.1초 마다 동작하고 쓰레드의 삭제 전 최대 대기시간 (idle timeout)은 0.3초로 설정하였다.

그림 13과 그림 14에서 본 실험의 결과를 그래프로 나타내었다. 실험 결과 제안하는 예측기반의 동적 쓰레드 풀이 워터마크 쓰레드 풀에 비해 응답시간이 길고, 메모리 사용량은 훨씬 적음을 볼 수 있다.

이는 워터마크 쓰레드 풀의 경우, 사용자 요청량이 적어 현재 동작중인 쓰레드의 개수가 낮은 워터마크 값 이하가 되어 고정 쓰레드 풀 모델과 같이 동작하므로, 일정한 개수의 워커 쓰레드를 풀에 대기시켜 사용자 요청에 응답 하므로, 응답시간이 빠르게 된다. 그러나 예측기반의 동적 쓰레드 풀의 경우, 사용자의 요청이 비연속적으로 오기 때문에 다음 쓰레드 량을 예측하여 미리

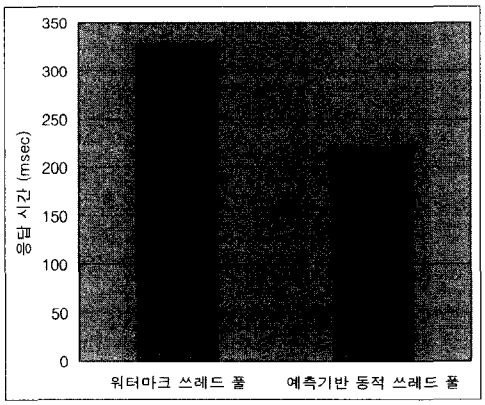

그림 11 높은 부하에서의 응답시간

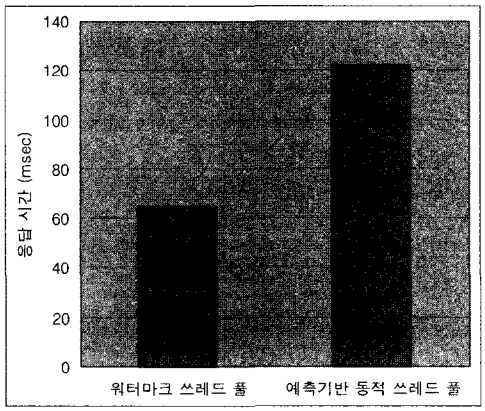

그림 13 낮은 부하에서의 응답시간

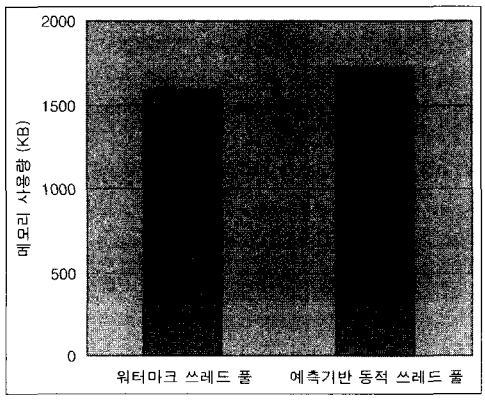

그림 12 높은 부하에서의 평균메모리 사용량

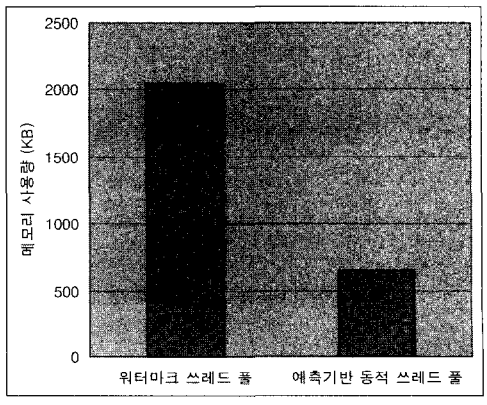

그림 14 낮은 부하에서의 평균메모리 사용량

생성하기가 어렵고, 또한 한번 생성된 쓰레드도 다음 요청이 오기 전에 삭제되어 재활용되지 않는 경우가 많아 다음 요청을 받으면 새로 쓰레드를 생성해서 처리하게 되므로, 고정 쓰레드 풀처럼 동작하는 워터마크 쓰레드 풀에 비해 응답시간이 늘어나게 된다.

그러나 메모리 사용의 측면에서 보면, 일정 개수의 쓰레드를 유지하는 워터마크 쓰레드 풀에 비해 쓰레드를 일정시간 대기시키긴 하지만 사용자 요청이 그 사이 없으면 쓰레드를 삭제하고 자원을 반납하는 예측기반의 동적 쓰레드 풀이 훨씬 더 적은 메모리를 사용하게 된다.

사용자의 서비스 요청이 적은 상황에서는 요청량이 많을 때보다 서버 시스템 자체의 부하가 적어 응답시간 빠르기 때문에 두 모델 사이의 응답시간 차이가 큰 의미가 없다. 오히려 제한된 시스템의 자원을 사용하여 여러 서비스를 해야 하는 임베디드 시스템이나 일반 범용 서버의 경우, 사용자의 요청이 적을 때 최대한 시스템 자원 반환하여 다른 프로그램에서 활용하도록 하는 것이 바람직한 시스템의 설계 방향이 될 것이다. 따라서 본 논문에서 제안하는 예측기반의 동적 쓰레드 풀 기법이 사용자의 요청량이 적은 상황에서 응답시간은 다소 늦더라도 시스템의 자원을 적게 사용하므로, 워터마크 쓰레드 풀 모델에 비해 요구하는 시스템의 설계 방향에 부합한 모델이라고 할 수 있다.

4.4 감시자 쓰레드의 부하 측정

본 실험에서는 감시자 쓰레드가 일정한 시간 간격을 두고 주기적으로 계속 동작하기 때문에, 반복 주기와 연산을 처리하기 위해 걸리는 시간 등에 따라 전체 서버 시스템의 성능에 영향을 미칠 수 있으므로, 감시자 쓰레드가 동작하는 간격에 따른 응답시간의 변화와 감시자 쓰레드의 연산시간을 측정하였다.

먼저 그림 10의 λ = 35의 부하를 감시자의 동작 시간 간격을 변경시켜가며 서버의 평균 응답 시간을 측정하고 그 결과를 그림 15에서 그래프로 나타내었다. 결과를 보면 감시자의 동작 간격이 사용자 요청이 생성되는 주기인 0.1초와 같을 때 응답시간이 가장 빠르다. 감시자 쓰레드의 동작 간격이 길어질수록 응답시간이 늘어났고, 0.1초 이하일 경우에도 응답시간이 늘어났는데, 이것은 사용자 요청의 도착 시간과 감시자 쓰레드의 검사 시간이 일치하지 않으면 현재 쓰레드 량의 산출이 덜 정확해 지므로 수정된 지수 평균을 이용한 예측 또한 덜 효율적이 된다. 따라서 보다 좋은 성능을 위해서는 사용자 요청의 도착 정도에 따라서 이와 비슷한 정도의 검사 시간을 설정하는 것이 정확한 예측을 통해 좀 더 효과적으로 동적 쓰레드 풀이 동작한다는 것을 알 수 있다.

마지막으로 감시자 쓰레드가 수행하는 지수평균을 구하기 위한 연산에 걸리는 시간은 240×10^{-6}~300×10^{-6}ms

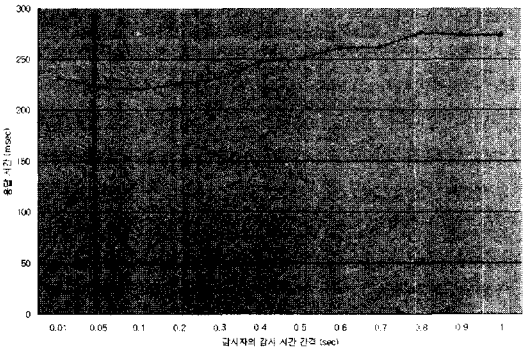

그림 15 감시자의 검사간격 변화에 따른 응답시간 변화

였다. 이는 쓰레드의 생성에 걸리는 시간인 20 ms에 비하면 훨씬 적은 시간이므로, 수정된 지수평균을 통해 필요한 쓰레드를 미리 예측하고 생성시키는 방법이 단순히 요청이 올 때마다 쓰레드를 생성시키는 방법보다 훨씬 효과적임을 알 수 있다.

5. 결론 및 향후 과제

워터마크 쓰레드 풀 모델의 경우 기존의 고정된 워커 쓰레드 풀 모델과 요구기반 쓰레드 모델의 단점들을 보완하여 쓰레드 풀을 크기를 사용자 요청량에 따라 동적으로 변하게 하는 모델로, 사용자 요청에 대한 응답시간은 적게 유지하면서 시스템의 자원을 효율적으로 활용하고자 개발되었다. 그러나 요청량이 많을 경우에 많은 사용자 요청에 따른 워커 쓰레드 생성 과부하에 의한 응답시간 지연과 사용자 요청량이 적을 때 불필요하게 시스템 자원을 할당하고 있는 부분은 여전히 존재하게 하는데, 이는 제한된 시스템 자원을 활용하여 서비스를 하는 자원의 효율적인 활용이 중요한 임베디드 시스템에서나 여러 종류의 서비스를 제공하는 범용 서버에서는 부적합하다. 따라서 필요한 쓰레드의 개수를 지수평균을 응용한 기법을 사용하여 예측하고 미리 생성해 놓음으로써, 사용자의 요청이 있을 때 쓰레드 생성으로 인한 응답시간의 지연을 개선하고, 사용자의 요청이 없을 때는 쓰레드를 삭제시켜 그 쓰레드의 자원을 다른 프로그램에서 사용할 수 있도록 하는 더 효율적인 구조로서 예측기반의 동적 쓰레드 풀 기법을 제안하고 실험을 통하여 그 성능을 확인하였다.

이를 통해, 사용자의 요청량이 많은 경우 시스템의 자원을 좀 더 사용하더라도 응답시간을 조금이라도 줄이는 것이 바람직하고, 요청량이 적은 경우에는 반대로 최대한 시스템 자원 반환하여 다른 프로세스에서 활용하도록 하는 것이 바람직한 시스템의 설계 방향이 될 것이므로, 본 논문에서 제안하는 예측기반의 동적 쓰레드

풀 기법이 요구하는 시스템의 설계 방향에 좀더 부합함을 알 수 있었다.

그러나 본 논문에서는 사용자의 요청량이 많은 경우와 적은 경우를 따로 실험하여, 전형적인 두가지의 상황을 놓고 기존 워터마크 쓰레드 풀 모델과의 비교를 실시하였다. 따라서 실제 상용 서버에서 사용량이 많은 경우와 적은 경우를 다 포함하도록 시간적으로 오랜 시간 성능을 측정하거나, 하나의 서비스를 하는 서버가 아닌 여러 서비스를 동시에 제공하는 일반 서버 환경에서 성능을 측정하여, 제안하는 예측 기반의 동적 쓰레드 풀 모델이 워터마크 모델보다 얼마나 효용 가치가 있는 모델인지를 보이는 방법에 대해서는 추후 연구가 더 필요하다.

참 고 문 헌

[1] D. Schmidt, I. Pyarali, M. Spivak, and R. Cytron, "Evaluating and Optimizing Thread Pool Strategies for Real-Time CORBA," ACM SIGPLAN Notices, Vol 36, No 8, pp. 214-222, August 2001.

[2] D. Schmidt and S. Vinoski, "Object Interconnections: Comparing Alternative Programming Techniques for Multi-threaded Servers-the Thread-Pool Concurrency Model," C++ Report, SIGS, Vol 8, No 4, April 1996.

[3] DynamicTAODocumentation, http://choices.cs.uiuc.edu/2k/dynamicTAO/doc/

[4] D. Schmidt, "Evaluating Architectures for Multi-threaded CORBA Object Request Brokers," Communications of the ACM, Special Issue on CORBA, ACM, edited by Krishnan Seetharaman, Volume 41, No. 10, October 1998.

[5] D. Schmidt and S. Vinoski, "Object Interconnections: Comparing Alternative Programming Techniques for Multi-threaded Servers-the Thread-per-Request Concurrency Model," C++ Report, SIGS, Vol. 8, No 2, February 1996.

[6] M. Welsh, S. Gribble, E. Brewer, and D. Culler, "A Design Framework for Highly Concurrent Systems," Technical Report UCB/CSD-00-1108, UC Berkeley, April 2000.

[7] J. Kurose and K. Ross, "Computer Networking: a top-down approach featuring the Internet," Addison Wesley, 2001.

[8] A. Silberschartz and P. Galvin, "Operating System Concepts," Addison Wesley, 1997.

[9] A. Concepcion, C. Stanton, and J. Torner, "Generic Tutorial System for the Sciences," http://gtss.math.csusb.edu/, 1997.

정 지 훈

2001년 서강대학교 컴퓨터학과 졸업. 2003년 서강대학교 컴퓨터학과 공학석사. 2003년~현재 LG전자 정보통신사업본부 재직중. 관심분야는 분산시스템, 운영체제 시스템

한 세 영

1991년 포항공과대학교 수학과 졸업. 2003년 서강대학교 정보통신대학원 공학석사. 2004년~현재 서강대학교 컴퓨터학과 박사과정. 관심분야는 분산시스템, 고성능 클러스터, 부하분산 시스템

박 성 용

1987년 서강대학교 전자계산학과 졸업 1994년 Syracuse 대학 Computer Science 석사. 1998년 Syracuse 대학 Computer Science 박사. 1998년~1999년 Bellcore 연구소 연구원. 1999년~현재 서강대학교 컴퓨터학과 부교수. 관심분야는 분산시스템, 클러스터 컴퓨팅 및 시스템, P2P 및 Grid 컴퓨팅