제 18 회 한국정보처리학회 추계학술발표대회 논문집 제 9 권 제 2 호 (2002.11)

자바 언어를 이용한 소켓폴링 서버구현

손강민, 강태근, 함호상
전자통신연구원 이동분산처리연구팀
e-mail : {sogairan, tgkang, hsham}@etri.re.kr

Implementing Socket Polling Server in Java

Kang-Min Sohn, Tae-Gun Kang, Ho-Sang Ham
Mobile Distributed Computing Research Team
Electronics and Telecommunications Research Institute

요 약

소켓 프로그래밍(socket programming) 인터페이스를 지원하는 C/C++, perl, python 과 같은 언어들은 폴링(polling) 기능을 갖는 select() 함수를 제공한다. 이 select()함수를 이용할 경우, 단일 쓰레드(또는 프로세스)로 다중의 클라이언트 요청을 처리할 수 있다. 최근 네트워크 프로그래밍 분야에서 주목받는 자바 언어의 경우, 최신 JDK 1.4 의 비동기 입출력 패키지에서 select()함수를 제공하고 있으나, JDK 1.3 을 포함한 그 이하의 버전에서는 아직까지 이 함수를 제공하지 않고 있다.

일반적으로 다중 쓰레드를 이용하여 소켓서버 응용프로그램을 개발할 경우, 코드가 단순해지고 응답이 빠른 장점이 있는 반면에 네트워크 연결이 증가할수록 다수의 쓰레드를 관리하는 일이 CPU 에 큰 부담이 된다. 반면에 소켓폴링(socket polling)을 사용할 경우, 이러한 연결 유지에 대한 부담이 줄어드는 대신, 다중 쓰레드를 이용하는 방법에 비하여 구현이 어렵다.

본 논문에서는 다양한 시뮬레이션 환경에서 세가지 소켓 프로그래밍 모델에 대하여 그 성능을 비교평가 하였다. 이 세가지 모델은 단순 다중 쓰레드 모델(typical multi-thread model), 단일 쓰레드 소켓폴링 모델(socket polling with single-thread model), 다중 쓰레드 소켓폴링 모델(socket polling with multi-thread model)이다. 본 논문에서는 다중 쓰레드 소켓폴링 모델을 제안하고 JDK 1.3.1 을 이용하여 구현하였다. 이 모델의 경우 복잡한 구조에도 불구하고 단순 다중 쓰레드 모델와 유사하거나 더 나은 성능을 보여주었다. 또한 동일한 용량의 쓰레드 풀(thread pool)을 사용하더라도 단순 다중 쓰레드 모델보다 더 많은 수의 클라이언트를 수용할 수 있는 장점이 있다. 이러한 결과를 바탕으로 본 연구팀에서 수행중인 MoIM-Messge 서버의 네트워크 모듈로 다중 쓰레드 소켓폴링 모델을 적용하였다.

KEYWORDS : 다중 쓰레드, 소켓폴링, 자바, select, multi-thread, socket polling, java

1. 서론

C/C++, perl, python 등의 언어를 이용하여 전통적인 소켓 서버 응용프로그램을 개발할 경우, 두 가지 방법을 이용할 수 있다. 첫번째는 fork()함수를 이용하여 독립적인 자식 프로세스(child process)를 생성하거나 쓰레드 프로그램을 이용하여 각각의 클라이언트에 서비스를 제공하는 방법이다. [1][2] (본 논문에서는 이 모델을 단순 다중 쓰레드 모델로 명명한다) 이러한 방법은 프로그램 코드를 간단하게 하고 일정 수의 클라이언트가 접속하는 경우에는 빠른 서비스 성능을 보여준다. 그러나, 클라이언트의 접속 수가 늘어갈수록 다수의 프로세스나 쓰레드를 관리하기 위하여 작업 전환(task switching)이나 가상 메모리 할당(virtual memory allocation)과 같은 일이 서버의 CPU 에 큰 부담으로 작용하여 속도가 느려지고 최악의 경우 동작 불능상태가 될 수도 있다. 이러한 문제는 쓰레드 풀(thread pool)을 사용하여 사용되는 최대 쓰레드의 수를 조절하여 막을 수 있다. 그러나 다중 쓰레드 모델에는 근본적으로 클라이언트로부터 데이터가 전송될 때까지 해당 쓰레드는 아무런 일도 하지 않고 공전(idle) 상태로 서버의 귀중한 시스템의 자원을 낭비하게 되는 문제가 존재한다.

두번째는 단일 쓰레드에서 select()함수를 이용하여 모든 클라이언트의 요청을 처리하는 방법으로, 보통 소켓폴링(socket polling)이라고 한다. 이 경우, 접속수와는 관계없이 단일 쓰레드를 사용하기 때문에 쓰레드 관리과 관련된 문제를 고려할 필요없이 다수의 클

라이언트를 수용할 수 있다. 그러나 서비스 품질(Quality of Service)의 관점에서 본다면, 접속하는 클라이언트의 수가 증가할수록 먼저 등록된 요청이 처리될 때까지 클라이언트가 기다리는 시간이 증가하게 된다. 또한 select() 관련 함수를 사용하여 서버를 설계하고 구현하는 것 자체가 fork() 함수를 이용한 다중 프로세스 또는 다중 쓰레드를 이용하는 단순 다중 쓰레드 모델보다 어렵고 복잡하다는 단점이 있다.

select() 함수의 경우 보통 네트워크 디바이스 드라이버의 I/O 인터럽트를 이용하여 구현되고 (대부분의 UNIX 머신에서 select()는 시스템 호출로 제공된다) JDK 1.3 까지 비동기 I/O 기능을 제공하지 않았기 때문에 자바에서는 select()함수가 제공되지 않았다. 현재 JDK 1.4 부터 java.nio 패키지에서 이러한 비동기 I/O를 제공하면서 이러한 문제가 해결되었다. 그러나 많은 자바 응용프로그램들이 아직도 JDK 1.3 을 기반으로 개발되고 있기 때문에 JDK 1.3 에서 select() 기능을 구현할 필요가 생기게 되었다.

대부분의 서버는 I/O 와 밀접하게 연관되어 있는데, 서버의 CPU 는 외부 장치로부터의 입력(소켓 서버의 경우에는 네트워크로부터의 입력)을 기다리며 99.9%의 시간을 공전상태(idle)상태로 놓여있게 된다. 이러한 문제는 서버와 클라이언트간의 대역폭 차이에 의해서 더욱 악화된다. [3] 단순 다중 쓰레드 모델을 이용하여 소켓 서버를 구현할 경우, 각각의 연결을 담당하는 쓰레드들은 각 소켓 스트림(stream)으로부터 데이터가 입력될 때까지 매우 긴 시간을 기다리게 될 것이다.

본 논문에서는 다양한 시뮬레이션 환경에서 세가지 소켓 프로그래밍 모델에 대하여 그 성능을 비교평가하였다. 이 세가지 모델은 단순 다중 쓰레드 모델(typical multi-thread model), 단일 쓰레드 소켓폴링 모델(socket polling with single-thread model), 다중 쓰레드 소켓폴링 모델(socket polling with multi-thread model)이다. 본 논문에서는 폴링 방법과 쓰레드 풀을 사용하여 JDK 1.3.1 에서 다중 쓰레드 소켓폴링 모델을 구현하였다. 이 모델의 경우 단순한 다중 쓰레드의 장점과 쓰레드 오버헤드 및 리소스 낭비를 막을 수 있는 장점을 제공한다. [4] 또한 복잡한 구조에도 불구하고 단순 다중 쓰레드 모델와 유사하거나 더 나은 성능을 보이고, 쓰레드 풀의 용량보다 더 많은 수의 클라이언트를 수용할 수 있다.

2. 세가지 소켓 프로그래밍 모델

본 논문에서는 단순 다중 쓰레드 모델(typical multi-thread model), 단일 쓰레드 소켓폴링 모델(socket polling with single-thread model), 다중 쓰레드 소켓폴링 모델(socket polling with multi-thread model)의 세가지 소켓 프로그래밍 모델을 제공한다. 단순 다중 쓰레드 모델의 경우 구현하기 쉽고 많은 소켓 서버에서 대부분 사용되고 있다. 단일 쓰레드 소켓폴링 모델은 MEDUSA [3] 에서 사용되고 있으며, 과부하 웹 서버에 적합하다. 본 논문에서는 다중 쓰레드 소켓폴링 모델을 제안한다. 이 모델은 앞의 두 모델을 조합한 것으로, 네트워크 연결 수에 영향을 받지 않으며 다양한 리소스 요구량에 맞도록 쓰레드 풀의 용량을 제어할 수 있다. (경우에 따라서 MEDUSA 스타일의 단일 쓰레드 모델로 설정할 수도 있다)

2.1 단순 다중 쓰레드 모델

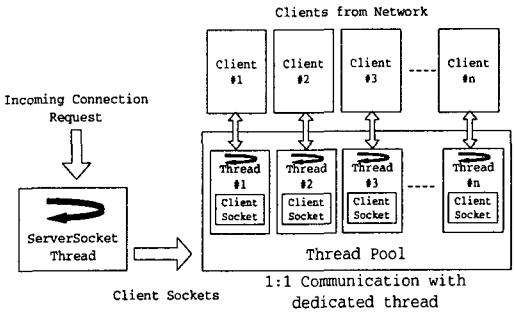

그림 1. 단순 다중 쓰레드 모델

그림 1 은 단순 다중 쓰레드 모델이다. 각 클라이언트 소켓은 전담 쓰레드에서 처리되므로 완료된 결과는 즉시 클라이언트로 전송된다. 반면에 클라이언트의 연결 수가 증가함에 따라서 동일한 수의 쓰레드가 생성되어야 한다. 그러나 대부분의 쓰레드들은 클라이언트로부터 요청이 발생할 때까지 리소스를 차지하며 공전상태로 남아있게 된다. 이러한 문제점은 다중 쓰레드 모델을 사용한 소켓 서버에 잠재되어 있는 결함이라 할 수 있다.

2.2 단일 쓰레드 소켓폴링 모델

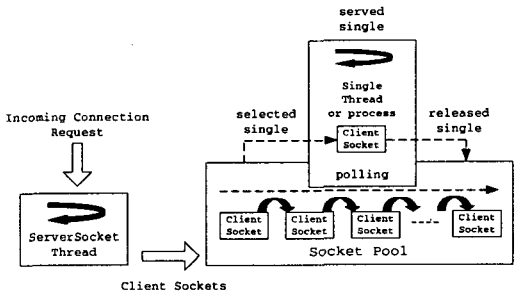

그림 2. 단일 쓰레드 소켓폴링 모델

그림 2 는 단일 쓰레드 소켓폴링 모델이다. 서버로부터 인가된 클라이언트 소켓은 소켓 풀(socket pool)에 저장된다. 단일 처리 쓰레드(single thread handler)는 폴링 과정을 통하여 데이터를 갖고 있는 소켓들을 순서대로 선택하고 요청을 처리한 후, 소켓 풀에 반환한다. 이 모델의 경우, 쓰레드보다 훨씬 부담이 적은 클라이언트 소켓용 메모리 공간만 증가하게 되므로, 클라이언트의 연결 수가 늘어나는 것에 큰 영향을 받지 않는다. 그러나 각 소켓이 자신의 요청이 처리되기까지 기다리는 시간 또한 늘어나게 되고, 이러한 대기 시간은 클라이언트와 서버간에 주고 받는 데이터의 용량에 매우 종속적일 수 밖에 없다. 따라서 단일 쓰레드

소켓폴링 서버를 구현하기 전에 주고받는 데이터 양의 고려가 우선적으로 선행되어야 할 것이다.

2.3 다중 쓰레드 소켓폴링 모델

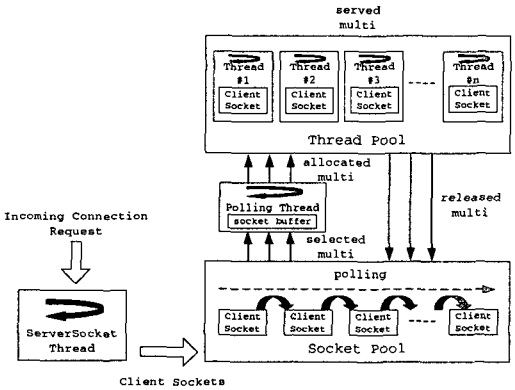

그림 3. 다중 쓰레드 소켓폴링 모델

그림 3 은 본 논문에서 제안한 다중 쓰레드 소켓폴링 모델이다. 이 모델은 앞의 단일 쓰레드 소켓폴링 모델과 유사하다. 여기에 폴링 쓰레드와 클라이언트의 다중 요청을 처리하는 쓰레드 풀이 추가되어 있다. 폴링 쓰레드는 소켓 풀로부터 데이터를 갖고 있는 소켓을 선택하여 쓰레드 풀에 할당하게 된다. 각 쓰레드는 데이터를 갖는 소켓의 요청을 한번만 처리하고, 처리 완료 후 즉시 소켓 풀로 해당 소켓을 반환한다. 나머지 소켓들도 등록된 순서대로 선택된다. 이 모델은 단일 쓰레드 모델과 마찬가지로 연결 수에 영향을 받지 않고, 단순 다중 쓰레드 모델처럼 다중 요청을 처리할 수 있다. 또한 쓰레드 풀의 용량을 조절함으로써 서버 자원의 사용량을 제어할 수도 있다.

폴링 스레드에는 소켓 버퍼(socket buffer)가 존재한다. 쓰레드 풀 내부의 쓰레드가 모두 사용중이 경우, 폴링 쓰레드도 공전상태에 놓일 수 있다. 그러나 소켓 버퍼의 도움으로, 폴링 쓰레드는 쓰레드 풀에 공급할 충분한 수의 소켓을 확보하는 작업을 멈춤없이 지속할 수 있다.

3. 자바에서 소켓폴링의 구현

JDK 1.3 에는 소켓폴링을 구현하는데 필요한 select() 함수가 제공되지 않으나, 몇 가지 함수를 조합하여 이를 모방할 수 있다. 자바언어에서 InputStream 은 C/C++언어의 파일 디스크립터(file descriptor)와 유사한 역할을 한다. 네트워크 장치를 포함한 외부 장치로부터 전송되는 모든 데이터들은 이곳을 통하여 전달된다. InputStream 에는 현재 수행 흐름을 방해하지 않고 한번에 읽을 수 있는 데이터의 양을 바이트 단위로 알려주는 available()이라는 함수가 제공된다. [5] 만약 읽어들일 데이터가 없는 경우에는 0 을 리턴하고, 그렇지 않은 경우에는 0 이 아닌 해당 데이터의 크기를 리턴한다. 이러한 특성을 이용하여 select()와 유사한 기능을 구현할 수 있다.

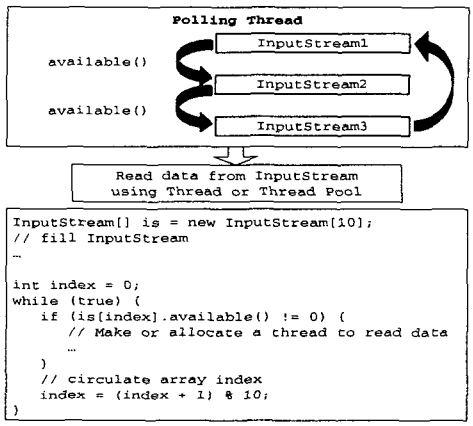

그림 4. 소켓폴링 모델과 자바 예제 코드

그림 4 는 단순화한 소켓폴링 모델과 자바 코드 예제이다. 먼저 ServerSocket 객체로부터 클라이언트 Socket 객체가 만들어지고, 다시 Socket 객체로부터 InputStream 객체를 얻을 수 있다. 이러한 InputStream 은 컨테이너 클래스(container class)인 배열(array)이나 연결 리스트(linked list)에 저장되어 소켓 풀을 형성한다. 폴링 쓰레드는 available() 함수를 이용하여 소켓 풀에 저장된 InputStream 을 조사한다. 만약 읽을 데이터가 존재한다면 해당 InputStream 의 레퍼런스를 쓰레드나 쓰레드 풀에 할당하여 read() 연산을 수행하게 된다. 이 read() 연산을 수행하는 동안에는 폴링 쓰레드의 조사 대상에서 제외되도록 해당 InputStream 를 "busy" 상태로 설정하여 중복 할당을 막도록 한다.

4. 시뮬레이션

본 논문에서는 세가지 모델에 대하여 각각 클라이언트 수, 메시지의 길이, 그리고 쓰레드 풀의 용량을 변화하며 실험을 수행하고, 그 수행 완료 시간을 결과값으로 기록하였다. 각각의 실험에 기본적으로 사용된 인수는 다음과 같다.

기본 실험 인수

클라이언트 수 : 100
메시지 수 : 100
메시지 길이 : 1024 (chars)
쓰레드풀 용량 : 1000

동작 환경

OS : Windows XP Professional
CPU : P-4/1.2GHz/640M
JVM : JDK 1.3.1

4.1 클라이언트 수

그림 5 는 클라이언트 수를 변화에 따른 수행 시간(로그값) 결과이다. 단일 쓰레드 소켓폴링의 경우 클라이언트 수가 1 일 때 가장 좋은 결과를 냈으나 나머지 실험에서는 모두 나머지 두 경우보다 안좋은 결과

를 보였다. 단순 다중 쓰레드 모델과 다중 소켓폴링 모델의 경우 유사한 결과를 보였으나 측정 데이터를 비교할 경우 다중 소켓폴링 모델이 약간 더 좋은 결과를 보이고 있다.

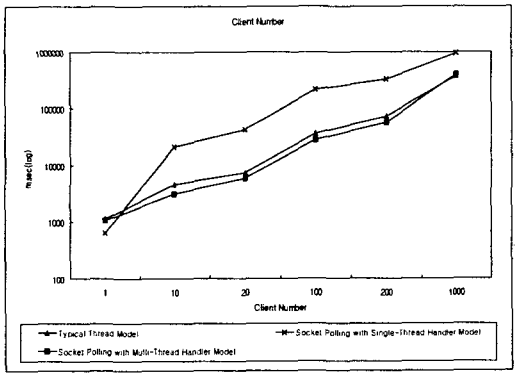

그림 5. 클라이언트 수 변화에 따른 수행 시간

4.2 메시지 길이

그림 6 은 메시지 길이 변화에 따른 수행 시간 (로그값) 결과이다. 본 실험에서는 512, 1024, 2048, 4096 문자로 변화하였으나 세 모델의 성능에는 별다른 영향을 미치지 않는다.

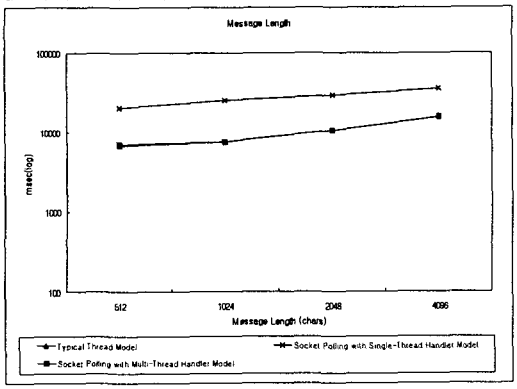

그림 6. 메시지 길이 변화에 따른 수행 시간

4.3 소켓폴링 모델의 쓰레드 풀 용량

그림 7 은 다중 쓰레드 소켓폴링 모델의 쓰레드 풀 용량 변화에 따른 평균 쓰레드 사용량과 수행 시간 결과이다. 비교를 위하여 다중 쓰레드 모델의 쓰레드 풀 용량을 100 으로 고정시키고 측정한 결과가 x 축과 평행하게 나타나있다.

그림에서 평균 쓰레드 사용량의 경우, 해당 쓰레드 풀의 용량과 거의 유사한 결과를 보이고 있다. 이는 쓰레드 풀의 용량을 거의 다 사용함으로써 효율적으로 할당된 자원을 사용하고 있다는 것을 알려준다. 여기에서 흥미로운 점은 쓰레드 풀의 용량이 20 인 시점부터 단순 다중 쓰레드 모델의 결과와 거의 유사하거나 더 좋은 결과를 보여준다는 것이다. 본 실험 결과에서는 30 의 쓰레드 풀 용량을 갖는 다중 쓰레드 소켓폴링이 모델 100 의 쓰레드 풀 용량을 갖는 단순 다중 쓰레드 모델과 동일한 성능을 보여주고 있다.

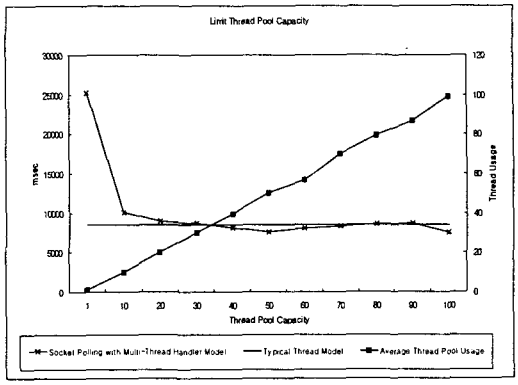

그림 7. 쓰레드 풀의 용량 변화에 따른 수행시간

5. 결론

소켓폴링은 오늘날 여러 프로그래밍 언어에서 소켓 서버를 구현하는데 사용되는 기법이다. 그러나 단일 쓰레드 소켓폴링 모델의 경우, 처리중인 클라이언트의 요청이 완료될 때까지 많은 클라이언트가 기다려야 하므로 요구 환경에 따라서 그 사용이 제한될 수 밖에 없다.

본 논문에서 제안한 다중 쓰레드 소켓폴링 모델의 경우 단순 다중 쓰레드 모델과 대등한 결과를 보이나, 쓰레드 풀의 사용 효율 측면에서 월등한 성능을 보인다. MoIM-Message 서버는 네트워크의 연결 유지에 관계없이 무선 클라이언트의 신뢰성있는 메시지의 송수신을 보장하는 이동 클라이언트 응용 서비스를 제공하는 메시지 미들웨어 시스템[6][7]으로 현재 한국전자통신연구원 이동분산처리 연구팀에서 개발중인 과제이다. 본 논문의 실험 결과를 바탕으로 다중 쓰레드 소켓폴링 모델은 현재 진행중인 MoIM-Message 서버의 네트워크 모듈로 구현되었다.

참고문헌

[1] Douglas E Comer, David L. Stevens "Internetworking with TCP/IP Vol.3: Client-Server Programming and Applications", Prentice Hall, 158~160

[2] Bil Lewis, Daniel J. Berg, "Threads Primer", Prentice Hall, 173~179

[3] Sam Rushing, "Medusa: A High-Performance Internet Server Architecture", rushing@nightmare.com, http://www.nightmare.com/medusa/medusa.html

[4] Jack Shirazi, "Java Performance Tunning", O'Reilly, 278~279

[5] Bruce Eckel, "Thinking In Java, 2nd Edition", Prentice Hall, 597~598

[6] A Fiorano White Paper, "A Guide to Understanding the Pluggable, Scalable Connection Management(SCM) Architecture in FioranoMQ5", Feb. 2001.

[7] "Benchmarking JMS Based E-Business Messaging Providers", Java Developer's Journal, Mar. 2001