클라이언트/서버 환경에서 클래스 이동을 고려한 자바 소스 코드의 시스템 부하 측정 시뮬레이터 구현

o
김 대현, 이 준연, 김 영찬
중앙대학교 컴퓨터공학과

Implementation of System Load Measuring Simulator of Java Source Code with Class Replacement in Client/Server Environment

o
DaeHyun Kim, JunYeon Lee, YoungChan Kim
Dept. of Computer Science and Engineering, Chung-Ang Univ.

요 약

클라이언트/서버 환경에서 성능향상을 위해서는 네트워크의 속도 향상, 서버와 클라이언트의 충분한 하드웨어와 운영체제, 데이터 베이스의 구성, 애플리케이션의 디자인을 고려해야 한다.

하드웨어에 추가되는 비용 없이 성능향상을 하기 위해서는 최적의 애플리케이션을 사용해야하는데, 코딩단계에서 개발 코드의 시스템 부하량을 측정하여 개발자에게 정보를 제공한다면 최적의 애플리케이션 개발이 용이할 것이다. 본 논문에서는 소스 코드의 정적 분석을 통해 시스템 부하량을 측정하는 시뮬레이터의 개발에 대해 논의한다.

1. 서론

클라이언트/서버 환경을 도입하는 이유는 개방형 시스템으로의 표준화와 자원공유, 확장성, 가용성, 애플리케이션의 생산성 등의 장점 때문이다.

이러한 점을 포함하며 클라이언트/서버 환경의 더 나은 수행 성능을 위해서는 다음의 요소들을 고려해야 한다

하드웨어와 관련하여 네트워크의 속도향상, 서버와 클라이언트의 시스템 업그레이드와 소프트웨어에 관련하여 운영체제, 데이터 베이스의 구성, 미들웨어를 포함한 애플리케이션 디자인 등이다.

하드웨어에 추가되는 비용 없이 성능향상을 하기 위해서는 주어진 환경을 최대한 활용하면서 클라이언트 또는 서버의 한쪽 호스트에 부하가 치중되지 않으며 두 호스트간의 네트워크 환경도 고려한 균형 있는 클리이언트/서버 애플리케이션을 개발해야 한다.[1][2]

클라이언트/서버 애플리케이션을 개발할 때 런타임에 애플리케이션을 시험을 한 후, 부하를 조정하기에는 비용이 너무나 크다. 이를 코딩 단계에서 개발 코드의 시스템 부하량을 측정하여 개발자에게 정보를 제공한다면 부하량 조정이 용이할 것이다

본 논문에서는 개발자가 최적의 클라이언트/서버 애플리케이션을 개발할 수 있도록 코딩 단계에서 소스 코드의 정적분석을 통해 부하량을 측정할 수 있는 시뮬레이터의 개발에 대해 논할 것이다.

2. 관련연구

2.1 애플리케이션 계층 구조

클라이언트/서버 환경은 클라이언트와 서버 사이에 데이터와 기능을 할당하는 방법에 있어 사용자와 개발자에게 상당한 유연성을 제공한다. 애플리케이션 구조의 범위와 복잡성은 호스트의 수, 데이터베이스의 크기, 데이터의 양, 동시 데이터 접근의 양에 의해 결정된다. 클라이언트/서버 애플리케이션은 본질적으로 표현(Presentation)계층, 애플리케이션(Application, Business Rules/Logic)계층, 데이터 계층의 세 가지 계층으로 나눌 수 있다.[1][2]

표현계층은 전체 시스템에 대한 사용자 인터페이스를 제공하는 계층으로 전체 시스템의 복잡성을 숨겨주며 입력 값과 선택사항을 편집하기 위하여 유효화와 검증로직 등을 필요로 한다. 애플리케이션 계층은 애플리케이션이 수행하여야 할 정책과 제약사항을 처리하는 계층으로 데이터, 상태, 이벤트-프로세서 모델에 대한 업무 규칙의 집합을 처리한다. 애플리케이션 계층은 하나 혹은 그 이상의 모듈로 나뉘어질 수 있으며, 클라이언트나 서버 혹은 양쪽으로 나뉘어 모두 위치될 수도 있다. 데이터 계층은 데이터베이스에 대한 트랜젝션과 질의는 이 계층에서 관리되고 접근(Accessing), 조회(Matching), 갱신(Updating), 삽입(Inserting), 저장(Retrieving), 백업(Backup), 복구(Recovery), 잠금(Locking), 보안(Security), 로깅(Logging)의 작업을 포함한다. 이러한 작업 중에서 몇 가지는 DBMS에 의해 수행되는데 클라이언트/서버는 적당한 물리적 장소에 대해 모듈화된 형식으로 이러한 계층들을 구성한다. 이것은 성능 문제상 요구되는 경우에 장소 변화에 대한 유연성을 제공한다.[3][4]

2-계층에서는 표현계층과 애플리케이션 계층은 클라이언트에서 함께 실행될 수 있기 때문에 패트-클라이언트(fat-client) 또는 클라이언트 중심(client centric)이라고도 한다. 이것은 많은 다른 애플리케이션에 비해 비용면에 있어서 효과적이고 상대적으로 간단한 구조이다.

이에 비해 확장성이 필요하고, 복잡한 물리적 구조는 3개이상의 계층을 이용한다. 애플리케이션 계층은 데이터베이스 서비스의 일부인 서비에 기반을 둔 애플리케이션 계층을 사용하기 보다는 새로운 계층에 놓음으로써 데이터베이스 서비스로부터 분리되어 사용된다.[5][6]

2.2 자바 가상 기계의 힙과 핸들

2.2.1 힙 (Heap)

힙은 자바 가상 기계에서 객체와 배열을 저장하는데 사용하는 메모리 블록으로서 자바 가상 기계는 객체와 배열을 불연속적으로 힙의 메모리 공간에 할당한다. 할당된 메모리는 객체나 배열이 가비지로 모아질 때 비워지게되고 자유 공간으로 표시된 후 다른 객체가 재 사용될 수 있게 된다.[7]

객체는 힙에 필드와 메쏘드의 두 부분으로 저장된다. 필드는 해당 자료형의 값을 저장할 메모리를 요구하며 메쏘드는 인수와 리턴값, 코드를 저장할 메모리를 요구한다. 메쏘드의 경우 메쏘드가 호출되는 경우에만 메모리를 할당받게 된다.

새로운 객체와 배열이 생성될 경우 힙에 메모리가 할당되며 이때 new 연산자는 객체나 배열이 저장되는 힙의 메모리 주소를 나타내는 레퍼런스를 리턴값으로 넘긴다.

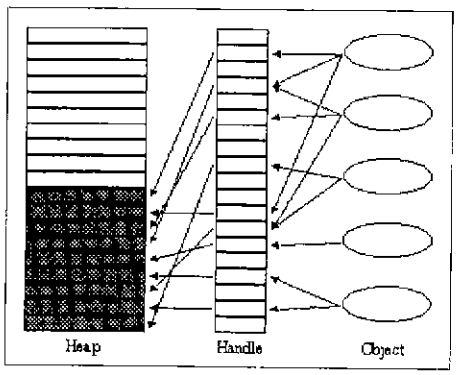
가상 기계는 힙을 관리하며 동일 메모리 블록이 동시에 두 개의 다른 객체 또는 배열에 할당되지 않도록 한다. 기본적으로 시스템에 따라 힙의 크기는 다르며 수행시 더 많은 힙이 필요할 때는 -ms<number>, -mx<number>의 옵션을 주어 크기를 늘릴 수 있다.

대부분 힙의 크기는 시스템이 사용할 수 있는 물리적 또는 가상적 메모리보다 훨씬 작다. 힙이 모두 사용된 경우에는 실행시 사용자에게 OutOfMemoryError 오류 메시지를 출력한다.

2.2.2 핸들 (Handle)

핸들은 포인터를 가리키는 포인터로서 실제 데이터가 저장된 주소가 메모리에 저장되어 있는 경우 메모리의 주소를 가리킨다.

핸들의 장점은 프로그램 수행시 핸들이 객체를 메모리 내에서 이동시키고 그 객체를 가리키는 포인터를 갱신할 수 있다는 점이다. 프로그램 수행중 힙에 객체나 배열을 불연속적으로 할당하게 되므로 다음 객체를 저장하기 위한 메모리는 충분하지만 힙의 메모리가 나뉘어지는 메모리의 조각(fragmentation) 현상이 일어나므로 메모리 블록을 이동시켜야하는 문제가 발생한다. 이 과정에 객체가 사용하는 포인터가 많고 다른 객체의 레퍼런스도 포함하는 복잡한 구조를 갖는 경우라면 객체와 메쏘드, 쓰레드에 있을 수 있는 모든 포인터를 수정해야하는 어려움이 생긴다. 이를 수정할 때의 오류와 수정을 하는 동안의 시간은 프로그램 수행 효율에 큰 영향을 끼치게 된다.



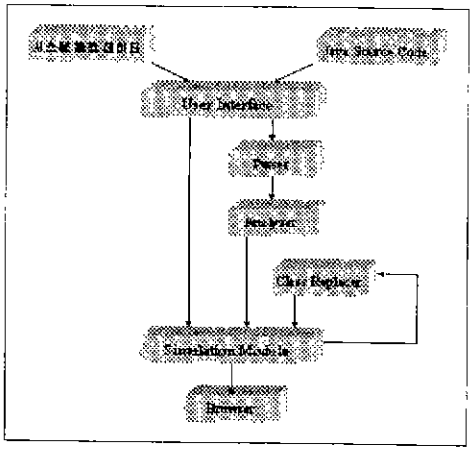
<그림 1> 핸들을 이용한 힙

이를 해결하기 위하여 <그림 1>처럼 힙과 같은 크기를 사용하는 레퍼런스 블록을 객체와 힙 사이에 넣는다. 힙에서 객체가 이동될 경우, 오프셋 테이블과 힙의 데이터 사이의 링크만 수정하면 된다. 오프셋 테이블과 객체 사이의 포인터는 수정할 필요가 없을 뿐 아니라 테이블 내의 수정이 필요한 포인터를 찾는 것도 비교적 쉽다.

오프셋 테이블의 사용중인 목록은 하나의 객체만을 가리키므로 같은 객체에 대해서는 포인터를 변경할 필요가 없다.

3. 시스템 구성 및 특성

<그림 2>는 시스템의 전체 구성도이다. 개발자가 자바 언어로 작성한 애플리케이션을 컴파일 하지 않은 정적인 상태에서의 시스템 부하를 측정하기 위한 본 시스템은 크게 사용자 인터페이스, 파서, 분석기, 클래스 리플레이서, 시뮬레이션 모듈, 브라우저의 6개 모듈로 구성되어 있다. 각 모듈의 특성은 다음과 같다



<그림 2> 시스템 구성도

3.1 사용자 인터페이스

먼저 개발자는 2-계층 시스템에서 수행될 애플리케이션을 시뮬레이션할 것인지 혹은 3-계층(또는 n-계층) 시스템에서 수행되는 애플리케이션을 시뮬레이션할 것인지를 결정한다.

각 계층에 해당하는 자바 소스코드와 애플리케이션이 수행될 시스템의 CPU 타입, 메모리 용량, 네트워크의 대역폭등의 데이터를 입력받는다

3.2 파서

입력된 자바 소스코드를 토큰으로 분리한다.

소스코드는 문법적인 오류는 없어야 하지만 컴파일 되기 전의 상태이므로 의미적(semantic)인 오류의 검사는 요구하지 않는다. 파서는 소스코드에서 시스템 부하 요소가 될 수 있는 요소로써 객체, 변수(지역, 매개)와 멤버 변수, 메쏘드를 추출한다. 추출된 요소들은 클래스와 클래스 단위로 정보가 저장되어 분석기에 전달된다.

3.3 분석기

파서에 의해서 추출된 정보를 시스템 부하로 정량화 시키는 역할을 한다.

JDK 내에 포함되어 있는 java_g는 jdb와 같은 디버거의 사용에 적합한 java의 최적화 되지 않은 버전이다. java_g는 "-t" 또는 "-tm"의 옵션과 같이 사용하여 인스트럭션과 메쏘드의 트레이싱을 할 수 있다. 소스 코드에 개발자가 만들었거나 사용한 메쏘드를 포함하여 트레이싱을 통해서 실제로 사용되는 모든 메쏘드를 찾는다. 이를 바탕으로 소스코드가 런타임시에 사용할 메쏘드의 시스템 부하를 예측한다.

시스템 환경 데이터로 입력받은 CPU 타입의 처리속도, 메모리 용량, 네트워크 속도를 변수로 시스템 부하량을 계산한다.

3.4 클래스 리플레이서

클라이언트/서버 환경에서 수행되는 애플리케이션이 최적의 성능을

발휘하려면 각 시스템 성능과 환경에 맞는 코드가 수행되어야 한다.

과서에 의해서 파싱된 코드는 클래스 별로 분석이 되어 각각의 시스템 부하 정보를 갖게된다 클래스 리플레이서는 각 계층별로 클래스를 표시하고 사용자의 선택에 따라 클래스의 부하를 이동시켜 게 시뮬레이션을 수행한다. 사용자는 이를 통해 개발 코드의 시스템 부하를 측정할 수 있고 클래스의 이동에을 시뮬레이션에 반영할 수 있다.

3.5 시뮬레이션 모듈

시뮬레이션 모듈에서는 분석기를 통해서 나온 정량화된 데이터를 바탕으로 모의시험을 수행한다

사용자에게 입력을 받았다 하더라도 네트워크의 상태는 애플리케이션이 수행되는 시간마다 크게 다를 수 있다. 네트워크의 상태가 좋지 못할 경우 클라이언트, 서버의 능력과는 상관없이 응답시간(response time)은 큰 차이를 보이게 된다. 본 연구에서는 이러한 상태의 변화를 랜덤 샘플링 과정에서 정규분포 확률 변수 값을 발생시키는 알고시즘을 이용하여 적용하였다. Marsanglia와 Bray는 Box와 Muller의 역변환 알고리즘을 변형하여 근사적으로 확률 변수를 발생시키는 균일분포의 합성방법을 제안하였는데, 이는 합성된 원소의 수가 증가되면 그 합은 정규 분포를 띠르게 된다는 중심 극한 정리에 의한 것이다.

3.6 브라우저

브라우저는 시뮬레이션 결과를 사용자에게 표시하는 역할을 한다 꺾은선 그래프와 막대 그래프는 각 계층별로 입력된 소스코드의 시스템 부하를 표시하고 원 그래프는 전체 응답시간에 대한 각 계층에 사용된 시간을 비례로 표시한다.

시뮬레이션 모듈의 결과치가 변함에 따라 응답시간을 동적으로 보여주며 전체 평균치도 표시한다

4. 수행 결과

일반적인 애플리케이션의 성능에 영향을 미치는 메모리의 양과 하드디스크의 속도를 포함한 기상 메모리의 양은 자바 애플리케이션의 수행 속도에는 큰 영향이 없다.

다음 <표 1>에 나타난 것처럼 일반적인 애플리케이션인 경우 메모리의 용량이 적더라도 수행 속도는 차이가 없거나 오히려 좋아지기도 하지만 <표 2>처럼 수행량이 방대한 경우 힙의 유지를 위한 가비지 콜렉션(garbage collection)등에 의해 수행 속도가 늦어진다. 본 연구에서는 메모리량에 따른 수행 속도 차이는 코드량별로 가중치를 두어 계산하였다.

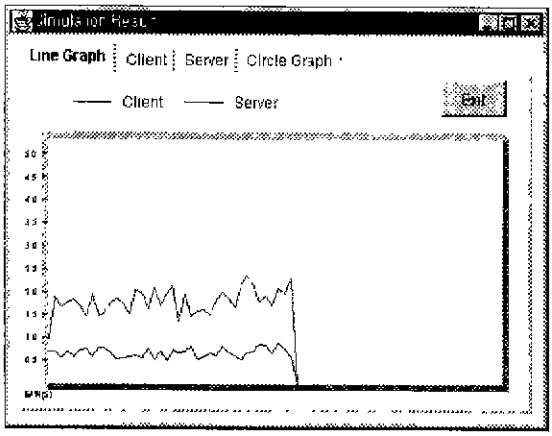
	1	2	3	4	5	평균(ms)
64Mb	2310	2250	2200	2310	2360	2286
32Mb	2090	2120	2290	2150	2220	2174

<표 1> 프로그램 A 수행시 응답시간

	1	2	3	4	5	평균(ms)
64Mb	30720	30120	29840	31560	30950	30638
32Mb	32020	31980	33260	32600	33190	32610

<표 2> 프로그램 B 수행시 응답시간

시스템 부하량에 가장 큰 영향을 주는 요소는 소스 코드내에 사용된 메쏘드들이다. 메쏘드가 갖는 부하량을 측정하고 소스 코드에서 추출된 정보를 정량화 시키기 위해 파서에서 검출한 메쏘드의 java_g에서 트래이싱한 메쏘드와 비교하였고 P200-MMX(64Mb 메모리)과 P133(32Mb 메모리)의 시스템을 기준으로 수행하여 얻은 수치를 기반으로 시스템 부하를 측정하였다. <그림 3>은 시뮬레이션 결과의 예를 보인 것이다.



<그림 3> 시뮬레이션 결과

5. 결론 및 향후 연구방향

시스템 부하량을 정확히 측정하기 위해서는 런타임에 모니터링을 통한 검사를 해야하지만 코딩단계에 개발자에게 시스템 부하 정보를 제공하기 위해서는 소스 코드를 분석하는 정적인 방법이 필요하다

본 시스템은 소스 코드가 수행될 시스템에 맞는 부하량을 측정하기 위해 수행 환경을 가정한 상태에서 시뮬레이션을 하고 결과를 사용자에게 보여준다 소스 코드내의 클래스가 이동한 경우로 재 시뮬레이션을 한 결과는 사용자에게 부하량을 조절하는 유용한 정보를 제공한다.

소스 코드를 분석하는 정적인 방법은 반복문과 제어 분기문을 포함하는 블록에 대한 사용 여부 및 빈도를 측정하기 어렵다. 이를 보안하기 위하여 확률변수를 발생하여 시뮬레이션을 수행하는데 이 과정에서 오차가 발생한다. 정적인 방법에서 생기는 오차를 줄이기 위해서 자바의 인터프리트적 특성을 이용한 리플렉션(reflection) 즉, 자바 클래스로부터 클래스 내부의 정보를 얻거나 코드의 커버리지를 검사하기 위한 최소한의 동적인 기법을 이용하면 좀 더 정확한 결과를 얻을 수 있을 것이다

6. 참고 문헌

[1] Waynew Eckerson, "Three-Tier Client/Server Architecture", 1995

[2] Johnson M. Hart, Barry Rosenberg, "Client/Server Computing for Technical Professionals", Addison-Wesley 1995

[3] Shaku Atre, "Distributed Database, Cooperative Processing & Networking", McGraw-Hill, 1993

[4] Steven Guengerich, "Downsizing Information Systems", SAMS, 1992

[5] Hisao Kameda, "Optical Load Balancing in Distributed Computer Systems (Telecommunication Network and Computer Systems)", 1997

[6] Andrew S. Tanenbaum, "Distributed Operating Systems", Prentice-Hall International Inc. 1995

[7] Elliotte Rusty Harold, "Java Secrets", 1997