그리드 컴퓨팅을 위한 프로세스 원격수행 모델

변상선, 진현욱, 고영웅, 유혁
고려대학교 컴퓨터학과
e-mail:{ssbyun, hwjin, yuko, hxy}@os.korea.ac.kr

The Remote Process Execution Model for Grid Computing

Sang-Seon Byun, Hyun-Wook Jin, Young-Ung Ko,
Hyuck Yoo
Dept. of Computer Science & Engineering,
Korea University

요약

그리드 컴퓨팅 환경을 이용하여 인터넷을 통해 프로세스를 원격에서 수행, 결과를 전달받을 수 있다. 기존의 프로세스 원격수행 모델은 다음과 같은 조건을 필요로 한다. 첫째, 해당 프로그램 또는 객체가 원격 컴퓨터에 존재하여야 하거나, 클라이언트와 서버관계를 설정하는 미들웨어가 요구된다. 둘째, 원격 프로그램 수행요청을 받아들이는 어플리케이션 또는 미들웨어가 미리 존재하여야 한다. 그러나 이와 같은 필요조건은 그리드 컴퓨팅 환경을 통해 계산자원을 동적으로 확장시키는데 사용되는 것을 제약한다. 본 논문에서는 이러한 필요조건 없이 원격 수행을 가능케 하는 프로세스 원격 수행 모델과 펑션 메시지를 활용하여 원격 계산자원을 동적으로 획득하는 방법을 제시한다..

1. 서론

그리드 컴퓨팅(Grid Computing)은 P2P (Peer-to-Peer)와 더불어 인터넷을 통해 WAN 구간에 분산되어 있는 계산 자원을 활용하여 분산컴퓨팅 및 병렬처리를 수행하는 기술이다[1][2]. 이 그리드 컴퓨팅은 기존의 클러스터링과는 달리 지리적으로 멀리 떨어져 있는 컴퓨터들을 활용하여 대규모 병렬 계산을 수행하는데 그 목적을 갖는다. 이러한 컴퓨터들을 계산에 참여시키기 위해 모빌 코드(Mobile Code)나 분산 객체를 사용할 수 있다.

기존의 원격 컴퓨터의 계산자원을 활용하는 시스템으로는 CORBA[3], RMI[3], DCOM[4], SOAP[4] 등의 분산객체를 활용하는 방법이 있으며, 이들은 사용하고자하는 객체의 메소드가 원격지의 컴퓨터 내에 있어야 하고 이 객체의 요청과 파라미터의 송수신을 위한 전용의 런타임 시스템이 요구되어진다. 이외에, 그리드를 위한 태스크 병렬 수행 모델인 Nexus의 경우도 원격 컴퓨터에서 수행될 코드와 파라미터의 송수신을 지원하는 런타임 시스템이다. Nexus는 원격 컴퓨터에서 실행될 코드를 인터럽트 핸들러로 등록을 하고, 액티브 메시지(Active Message) 기법을 사용하여 원격 컴퓨터에게 파라미터와 실행될 프로시저의 주소를 전달한다. 이는 파라미터와 프로시저 주소의 도착 사실을 인터럽트를 통해 알리고 인터럽트 핸들러로 수신 받은 주소에 위치한 프로시저를 수행하게 된다. 이런 형태로의 수행은 원격 컴퓨터가 멀티프로세서 환경일 경우 컴퓨터 내부에서의 데이터 병렬 처리에 제약을 가져온다[5].

원격 컴퓨터의 계산자원 활용을 위해 자바 애플릿[3], Safe-Tcl[6], Omniware[7] 등과 같은 모빌 코드를 사용할 수 있다. 이러한 모빌 코드는 웹브라우저나 가상기계를 구성하는 프로세스가 모빌 코드를 기다렸다가 받아서 수행해야한다. 자바 애플릿의 경우는 지역 디스크(local disk)의 접근이 불가능한 문제점이 있다. 이러한 특성은 그리드 환경에서 동적

으로 계산자원을 획득하는데 제한이 된다.

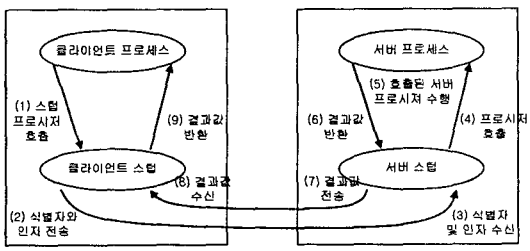
이와 같은 기존 시스템의 문제들을 해결하기 위해서 본 논문은 원격 컴퓨터의 계산 자원의 동적 획득을 위해 별도의 런타임 시스템을 필요로 하지 않고, 하나의 컴퓨터의 멀티프로세서 간에 데이터 병렬 처리를 이룰 수 있는 프로세스 원격수행 모델[8]을 그리드에 활용하는 방안을 제시한다.

본 논문은 다음과 같이 구성되어 있다. 서론에 이어 2장에서 평션 메시지와 관련된 기존의 연구를 살펴보고, 3장에서 평션 메시지의 설계 및 구현을 설명한다.

2. 관련 연구

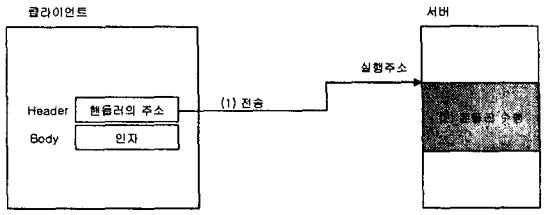
프로세스 원격 수행의 초기 형태는 원격 프로시저 호출(Remote Procedure Call)이나 엑티브 메시지(Active Message)라고 할 수 있다.

원격 프로시저 호출은 클라이언트가 서버에 존재하는 함수를 호출하는 방식으로 그 순서는 <그림 1>과 같다. 이러한 원격 프로시저 호출의 대표적인 예로는 CORBA, DCOM, SOAP, RMI등의 분산 객체지향 모델이 있다. 클라이언트는 프로시저의 식별자(identifier)와 인자를 메시지로 전송하면 서버가 식별자에 해당하는 프로시저를 수행하고 결과 값을 다시 클라이언트에게 전송한다. 이 때, 서버에는 해당 프로시저를 수행할 프로세스가 등록되어 있어야 한다. 등록되지 않은 임의의 프로그램은 수행될 수 없으며 프로그램 자체가 서버에 물리적으로 저장되어 있어야 한다.



<그림 1> 원격 프로시저 호출 수행 모습

클러스터링 환경에서 높은 성능을 얻기 위해 제안된 엑티브 메시지는 전송 측이 수신 측의 실행 위치를 지정할 수 있다. 즉, 각각의 메시지는 메시지가 도착했을 때 수행되어야할 핸들러의 주소를 헤더에 저장하고, 핸들러의 인자들을 메시지의 내용으로 삼는다. <그림 2>에서 클라이언트는 핸들러 주소를 헤더에 넣고, 인자들을 메시지의 내용으로 하여 서버에게 전송한다. 서버는 메시지를 받았을 때 메시지 헤더의 주소가 가리키는 핸들러를 수행시킨다[9].



<그림 2.> 액티브 메시지 수행 모습

공통적으로 위의 두 프로토콜은 클라이언트에 의하여 요구되는 기능을 서버가 미리 준비해야 하고 클라이언트에서는 서버가 어떤 기능을 수행할 수 있는지 알고 있어야 한다. 즉, 서버는 클라이언트로부터 요구되는 임의의 프로그램을 수행 할 수는 없으며 클라이언트도 서버가 제공하는 기능만을 요청할 수 있다. 따라서 원격 프로시져 호출과 액티브 메시지는 동적으로 원격 컴퓨터의 계산자원을 획득하는데 제한이 된다. 또한, 액티브 메시지는 인터럽트 핸들러의 일부로 코드가 수행되기 때문에 멀티프로세서에서 데이터 병렬처리를 이룰 수 없다. 서버가 클라이언트의 요구를 미리 준비하고 있는 구조는 동적인 계산자원 확장과, 데이터 병렬처리에 부적합하다. 또한, 클라이언트는 서버의 기능이 무엇인지 고려할 필요가 없어야 된다. 그리고, 기존 시스템의 변경이 없이 새로운 기능의 추가가 가능해야 한다.

3. 평션 메시지의 설계 및 구현

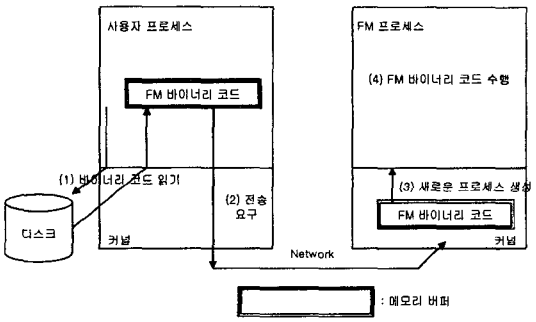
3.1 설계

이 수행 모델은 세 가지의 특징을 갖는다. 첫째, 모빌 코드를 요구하거나 모빌 코드를 수행하기 위하여 기다리는 프로세스가 존재하지 않는다. 그리하여, 별도의 가상기계나 런타임 시스템이 있어야 하는 현재 수행 모델의 한계점을 극복한다. 둘째, 전송된 모빌 코드는 메시지로서 전송되며, 이를 받는 원격 컴퓨터의 커널은 새로운 프로세스를 생성하여 모빌 코드를 수행시킨다. 셋째, 기존의 모빌 코드가 갖는 갖는 지역 디스크 접근과 같은 한계점을 제거한다. 이러한 수행 모델을 지원하기 위하여 본 논문에서 사용하는 모빌 코드는 컴파일되어 수행 가능한 바이너리 코드이며, 이것을 평션 메시지(Function Message)라고 부른다. 평션 메시지는 그 자체가 바

이너리 코드이기 때문에 수행되어져야 할 프로그램이 원격 컴퓨터 디스크에 존재할 필요가 없다. 그리고, 원격 컴퓨터에서 새로운 프로세스를 생성하여 프로그램을 수행하기 때문에 기존의 실행환경이 파괴되지 않는다. 또한, 액티브 메시지와 달리 실행 코드를 수행하는 프로세스를 생성하기 때문에 멀티프로세서 컴퓨터 내부에서 데이터 병렬처리를 이룰 수 있다.

평션 메시지의 동작은 다음과 같다. 첫째, 클라이언트의 디스크에 있는 바이너리 코드를 읽어서 평션 메시지를 만든다. 둘째, 이 메시지를 네트워크로 전송한다. 셋째, 평션 메시지를 받은 컴퓨터의 커널은 새로운 프로세스를 만든다. 넷째, 프로세스가 평션 메시지의 바이너리 코드를 수행한다. 이러한 순서가 <그림 3>에 나타나 있다.

평션 메시지에서 메시지를 전송 받는 것은 서버에 있는 프로세스가 아니라 서버의 커널이며, 전송받은 바이너리 코드가 프로세스의 주소 공간을 구성하여 수행이 이루어지므로 작업을 수행하는 것은 커널이 새로 생성하는 프로세스이다.



<그림 3> 평션 메시지 수행 모습

평션 메시지에 있어서 어려운 점은 평션 메시지로부터 프로세스를 생성하는 것이다. 새로운 프로세스의 생성은 두 단계로 나누어진다. 하나는 주소 공간을 만드는 것이고, 다른 하나는 주소 공간을 채워 넣어 수행할 수 있는 상태로 만드는 것이다. 기존 UNIX와 같은 커널에서 주소 공간을 만드는 것은 fork 시스템 호출을 사용한다. fork는 부모 프로세스의 가상 주소 공간이나 레지스터 값과 같은 정보를 바탕으로 새로운 프로세스를 만들게 된다. 그러나 평션 메시지에서는 새로운 프로세스의 생성을 요구하는 부모 프로세스가 없기 때문에 임의의 프로세스를 선택하고 그 정보를 바탕으로 프로세스를 새로 만들어야 한다.

이처럼 주소 공간이 만들어지려면, 이제 평션 메시지에 있는 바이너리 코드로 주소 공간을 채워 넣어야 한다. 이 작업은 UNIX의 exec와 비슷하다. 그러나 한가지의 문제점은 exec에서 사용하는 방식을 그대로 사용할 수 없다는 것이다. 왜냐하면 exec은 디스크에 저장된 바이너리 코드를 읽어서 프로그램이 실행하게 되면 이 코드를 주소 공간에 채워 넣는데 이 때, 바이너리 코드의 경로를 이용하기 때문이다. 그러나, 평션 메시지에서는 바이너리 코드의 내용이 디스크가 아닌 메모리 상에 존재하므로 새로운 방식이 필요하다. 이 새로운 방식은 디스크의 바이너리 코드가 아닌 메모리 상의 내용으로 주소 공간을 채우므로 이름을 mexec(memory exec)라고 한다. 즉, 커널이 네트워크로 전송된 바이너리 코드를 버퍼에 저장하고 이 버퍼에 있는 내용으로 mexec를 수행한다.

설명을 쉽게 하기 위해 본 논문에서는 커널 내부 버퍼에 저장된 평션 메시지를 'FM 바이너리 코드'라고 하고, FM 바이너리 코드를 수행하기 위해 운영체제가 새로 만들어내는 프로세스를 'FM 프로세스'라고 한다.

3.2 구현

평션 메시지를 위해 생성되는 FM 프로세스는 초기 데몬 프로세스를 복사하여 얻어진다. 초기 데몬 프로세스는 시스템의 최초 사용자 프로세스로 모든 사용자 프로세스는 초기 데몬 프로세스의 직접 또는 간접적인 자식이 된다. 따라서 FM 프로세스도 초기 데몬 프로세스의 자식이 되어도 무방하다. 이렇게 생성된 프로세스의 주소 공간의 내용과 레지스터의 값을 FM 바이너리 코드의 정보에 따라 변경하기 위해 mexec을 수행한다. mexec의 인자는 다음과 같다.

```
int mexec(struct task_struct*, void* start,
size_t len, pt_regs* reg)
/* tsk : FM 프로세스 구조체
   start : FM 바이너리가 저장된 버퍼의 시작 주소
   len : 버퍼의 길이
   reg : FM 프로세스의 레지스터 값이 저장될 위치
   return value : 성공하면 0, 실패하면 -1      */
```

mexec의 구체적인 구현 내용은 다음과 같다.

(1) 새로운 페이지를 할당받아 FM 바이너리를 수행하기 위한 인자와 환경 변수를 설정/저장한다. 이

페이지들은 이후에 FM 프로세스의 사용자 스택을 이루게 된다.

(2) FM 바이너리 코드의 헤더를 분석한다.

(3) FM 프로세스 주소 공간의 불필요한 정보를 제거한다.(예 : 초기 데몬 프로세스 정보)

(4) FM 바이너리 코드의 내용을 새로운 페이지로 복사하고 이 페이지들을 FM 프로세스의 주소 공간으로 매핑하여, FM 프로세스의 수행 중 페이지 폴트가 발생하지 않도록 하는 것이다.

(5) 인자와 환경 변수가 저장된 페이지를 FM 프로세스의 주소 공간에 연결시키고, 스택 영역을 계산하여 인자, 환경 변수에 대한 주소를 저장한다.

(6) FM 프로세스의 처음 수행 위치 IP(Instruction Pointer)와 SP(Stack Pointer)를 계산하여 FM 프로세스의 구조체에 저장하고 FM 프로세스를 실행 큐(Run queue)에 넣는다.

(7) FM 프로세스는 다른 프로세스들과 마찬가지로 스케쥴러에 의해 선택되어 실행된다.

mexec 전과정을 통해 물리 주소를 사용하는데 가상 주소를 사용하게 되면, FM 프로세스가 현재 프로세스가 아닐 경우, 현재 수행중인 프로세스의 주소를 접근하게 되고, FM 프로세스가 현재 프로세스라 하더라도 fork에 의한 프로세스 생성이 아니기 때문에 커널 스택의 내용 보장이 안되기 때문이다 [10].

3.3 그리드를 위한 FM

이상과 같이 기술된 FM 기법은 그리드 환경에 적용되기 위해서 추가되어야 할 요소들은 다음과 같다.

(1) 분산 병렬 계산을 위해서 FM과 함께 데이터의 분산이 이루어져야 한다. 현재 FM은 계산에 사용되는 데이터들이 바이너리 코드 안에 함께 포함되어 있다. 따라서 동적인 데이터의 분배를 위해서는 계산에 사용되는 데이터를 바이너리 코드와 분리해야 한다.

(2) FM 프로세스가 생성되어 수행된 후에 일반 프로세스와 마찬가지로 종료 후, 결과 값에 대한 회신 기능이 추가되어야 한다.

(3) 각 시스템의 부하 정보를 수집/분석하여 공정한 부하 분배가 이루어져야 한다.

(4) 메모리 대신에 램드라이브를 사용함으로써 동적 링크로 컴파일된 바이너리 코드의 수행과 페이즈 스왑의 허용이 가능해야 한다.

(5) 네트워크 지연을 최소화 하기 위해 바이너리 코드의 크기가 축소되어야 한다.

(6) 시스템의 보안 및 플랫폼의 독립성이 제공되어야 한다.

4. 결론

펑션 메시지를 사용하면 특정 언어에 의존적이지 않으며, 사용자 프로세스를 새로 생성하여 코드를 수행하므로 커널에 영향을 미치지 않아 안전성을 갖는다

본 논문은 사용자가 원격 컴퓨터에게 수행할 작업에 해당하는 임의의 바이너리 코드를 전송하여 원하는 작업이 이루어지도록 하는 펑션 메시지를 활용, 동적으로 원격 계산자원을 획득할 수 있는 모델을 제시하고, 그리드에 적용하기 위해 요구되어지는 사항들을 고찰하였다.

참고문헌

[1] G. Fox and D. Gannon, "Computational Grids," Computing in Science & Engineering, Volume: 3, Issue: 4, p74-77, July-Aug. 2001.

[2] I. Foster, C. Kesselman, J. M. Nick and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," Technical Report: the Globus Project, Jan. 2002.

[3] http://java.sun.com /.

[4] http://www.microsoft.com .

[5] I. Foster, C. Kesselman and Steven Tuecke, "Nexus: Runtime Support for Task-Parallel Programming Languages," Technical Report: Mathematics and Computer Science Division, Argonne National Laboratory, 1994.

[6] N.Borenstein, "Email with a Mind of its Own: The Safe-Tcl Language for Enabled Mail," IFIP International conference, May 1994.

[7] Adl-Tabatabai, G. Langdale, Steven Lucco and Robert Wahbe "Efficient and Languate-Independent Mobile Programs," ACM SIGPLAN'96 Conf. on PLDI, May 1996.

[8] S.H. Jeon, M.H Lim and Chuck Yoo, "A Remote Execution Model for Mobile Code," IEICE Transactions on Information and Systems, Vol.E83-D, No.11, p1924-1930, Nov. 2000

[9] T. von Eicken, D.E. Culler, S.C. Goldstein, K.E. Schauser, "Active Messages: a Mechanism for Integrated Communication and Computation," Computer Architecture, 1992. Proceedings, The 19th Annual International Symposium p256-266, 1992.

[10] D. P. Bovet, M. Cesati, "Understanding the Linux Kernel," O'Reilly & Associates, Inc., 2001.