클러스터 기반 그리드 응용의 부작업 간 통신 구현 사례

정평재° 이윤석
한국외국어대학교 전자정보공학부
{system9°, rheeys}@dslab.hufs.ac.kr

조금원
한국과학기술정보연구원
ckw@hpcnet.ne.kr

Implementation of Inter-subjob communication for Grid applications on Clusters

Pyongjae Jung° Yunseok Rhee
Hankuk Univ. of Foreign Studies

Keumwon Cho
Korea Inst. of Science & Tech. Information

요 약

클러스터 자원이 그리드 환경에서 효율적으로 운영되기 위해서는 클러스터를 구성하는 노드 간의 부하 균형이 필요하며 이를 위해 실행 중 프로세스 이동은 당연히 지원되어야 한다. 그러나 대부분의 그리드 지원 환경에서는 작업 시작, 실행 중 동기화 과정 등에서 프로세스들의 실행 중 이동이 고려되지 않았으며, 프로세스의 이동을 지원한다 하더라도 TCP 소켓 이전 등의 오버헤드가 따른다. 본 연구에서는 클러스터 내의 프로세스들이 실행 중에 노드 간을 이동하더라도 상호 통신이 가능하도록 별도의 통신 지원 쓰레드를 설계하고, 이를 매개로 간접 통신이 이루어지도록 개선된 방법을 제안하였다.

1. 서 론

그리드(Grid) 시스템은 지리적으로 분산되어 있는 고성능 컴퓨팅 자원을 네트웍으로 연동시켜 조직과 지역에 구애됨이 없이 투명하고 편리하게 사용할 수 있는 컴퓨팅 환경의 제공을 목표로 한다[1]. 현재 Legion, Condor, Globus toolkit[2] 등의 여러 그리드 지원 시스템이 개발되고 있으며, 이 중 Globus 툴킷은 기존 시스템과의 협응성과 각 서비스의 독립성이 뛰어나 가장 널리 사용되고 있다.

그리드 어플리케이션은 대개 여러 자원들을 동시에 필요로 하는 복합 작업(compounded job)이고, 이 때 요구되는 모든 자원 요청은 동시 할당자(co-allocator)를 통해 대신 이뤄진다. 동시 할당자는 하나의 작업을 다수의 부작업(sub job)으로 나눈 후에, 각각의 부작업을 각 지역 자원관리자에게 전달하면, 이들은 해당 지역의 자원을 할당받고 실행 관리된다.

현재 Globus는 DUROC(Dynamically Updated Request On-line Co-allocator) [3]이라는 구성 요소를 통해 동시 할당 기능을 지원하며, 모든 자원 요구가 만족될 경우에만 전체 작업(즉 모든 부작업들의 집합)을 진행하는 원자성(atomicity)을 준수한다. Globus는 이와 같은 원자성 구현을 위해 작업 시작 배리어(job-start barrier)라는 개념을 사용한다 [3]. 이는 서로 다른 자원 관리자에게 분산된 각각의 부작업들이 자원을 할당 받아서 실행에 필요한 모든 준비 작업이 완료되면, 이 부작업들은 작업 시작 배리어 코드를 수행하면서, 다른 부작업들이 해당 지점에 모두 도착하기를 기다리다가 모든 부작업들이 배리어에 도착한 것이 DUROC에 의해 확인되면, 비로소 각 부작업들은 수행을 시작하도록 하는 것이며 이를 위해 부작업 간의 활발한 통신이 요구된다.

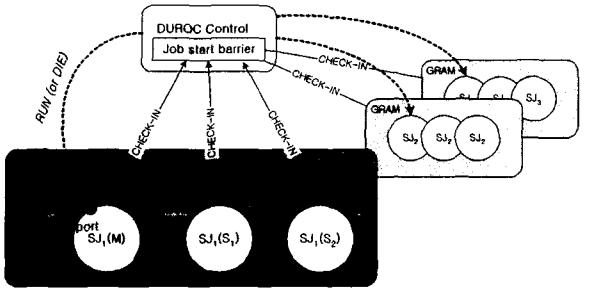
현재까지 개발된 대부분의 그리드 지원 시스템들은 원칙적으로 각 자원이 단일 시스템 또는 단일 자원 스케줄러를 기반으로 동작하고, 각 사이트에서 생성된 작업들이 논리적으로 일괄 관리될 수 있는 환경을 가정하고 개발되었다. 따라서 PC 클러스터와 같이 다수 컴퓨팅 노드로 구성된 자원 하에서 이들 사이를 프로세스들이 실행 중에 이동(migration)하는 환경에서는 앞서의 가정은 현실성이 없거나, SSI(single system image)를 제공하는 별도의 소프트웨어 계층을 요구한다. 그러나, 대부분의 그리드 어플리케이션이 MPI 기반인 점을 감안하면 SSI 미들웨어는 비용 대비 성능이 낮을 것으로 판단된다.

따라서 본 논문에서는 클러스터에서 실행 중 프로세스 이동이 발생해도, 프로세스 간 통신이 가능하도록 함으로써 그리드 어플리케이션의 동기화나 작업 시작 배리어가 동작할 수 있도록 개선된 부작업 간 통신 방법을 제안한다. 2절에서는 Globus DUROC과 지역 자원관리자 GRAM(Globus Resource Allocation Manager) [2] 시스템에서 그리드 응용 프로그램을 실행시키는 과정을 살펴보고, 3절에서 작업 이동이 가능한 PC 또는 웍스테이션 클러스터에서 작업 간 동기화를 지원할 수 있도록 개선한 설계 내용을 소개하고 결론을 맺는다.

2. 부작업 동기화 과정

Globus의 자원 동시 할당자(DUROC)은 작업 배리어 개념을 사용하여 각 지역에 제출된 부작업들의 동기화와 전체 작업의 원자성을 실현하고 있다. 이를 위해서는 각 부작업 간, 또는 부작업 내의 프로세스 간의 통신이 지원되어야 하는데, Globus에서는 이 기능들을 TCP 기반으로 구현하고 있다.

그림 1에 보이는 바와 같이, 하나의 부작업(sub-job)은



[그림 1] 그리드 응용의 작업 시작 과정

하나의 GRAM에 의해 관리되며, 각 부작업은 다수의 프로세스를 생성할 수 있고, 이 때 최초 생성 프로세스를 Master 프로세스, 그 외의 프로세스들을 Slave 프로세스들로 구분한다. 그림에 보이는 $SJ_1(M)$, $SJ_1(S_1)$, $SJ_1(S_2)$ 등은 부작업 1에서 생성된 Master와 Slave 프로세스들을 각각 나타낸다. 각 프로세스는 초기화 과정에서 자신의 준비 상태를 알리기 위해 DUROC 서버에게 진입(check-in) 메시지를 전달하며, DUROC 서버는 모든 프로세스들이 진입 상태에 도달하면 (또는 이에 실패하면), 각 GRAM의 Master 프로세스에게만 실행(RUN) (또는 중단(DIE)) 메시지를 전달한다. 그리고, 이를 전달받은 Master 프로세스는 자신이 관리하는 Slave 프로세스들에게 이미 수립된 통신 세션을 사용하여 동일 메시지를 동보 전송한다.

이와 같은 메시지 전달 과정이 성공하기 위해서는 각 프로세스 간에 이미 수립된 통신 세션이 지속적으로 유효해야 한다. 그러나, 클러스터 자원이 효과적으로 운영되기 위해서는, 전체 부하의 상태에 따른 작업 이동이 당연히 지원되어야 하는데, 이 과정에서 각 프로세스의 실행 노드가 동적으로 바뀔 수 있다. 결국 프로세스 생성 당시에 각 프로세스 간에 수립된 세션은 더 이상 유효하지 않게 된다. 따라서 이 구조는 클러스터 상에서 프로세스의 이동을 제약하거나 소켓 객체를 이동시켜야 하는 부담을 가져온다.

3. 클러스터에서의 부작업 동기화 과정 설계

우선 본 논문에서는 리눅스를 기반으로 한 PC 클러스터를 대상으로 구현 방안을 논의하지만, 다른 형태의 클러스터에도 큰 변경없이 적용될 것으로 생각된다.

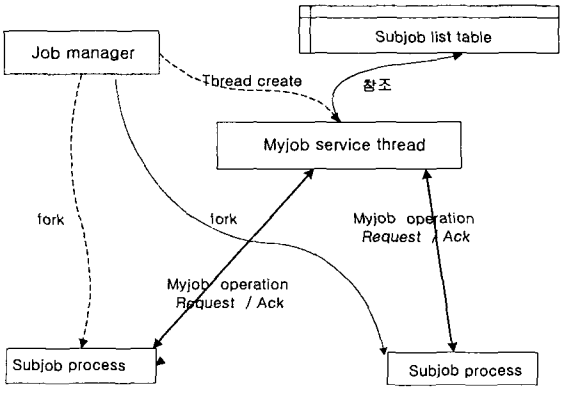
리눅스 클러스터에서 프로세스(또는 쓰레드) 이동을 지원하기 위해서 MOSIX 커널[4]을 채택하였으며, 그림 2에 보이는 바와 같이, 각 부작업에 대해 위해 프로세스를 생성하고 관리하는 작업 관리자(Job manager) 쓰레드, 이와는 별도로 프로세스 간 통신을 지원하는 Myjob 서비스 쓰레드(Myjob service thread)를 둔다. (참고로, 명칭 Myjob은 하나의 GRAM에 주어진 동일한 부작업 내의 프로세스들을 뜻하기 위해 사용한다.) 쓰레드 구현을 위해서는 POSIX pthread 라이브러리[5]를 사용하였다.

Myjob 서비스 쓰레드는 각 작업 관리자마다 별도로 필요하며, 하나의 부작업 내에 속한 모든 프로세스들 간의 통신을 지원한다. 기본 개념은 이 서비스 쓰레드를 매개로 각 프로세스 간에 간접 통신을 지원하는 것이다. 이 방법에서는 서비스 쓰레드의 위치를 고정시키고, 지정된 서비스 포트를 사용하도록 한다. 그리고 서비스 쓰레드는 내부에 각 프로세스 정보와 수신 메시지 큐를 갖춘 부작업 리스트 테이블(subjob list table)을 준비한다.

테이블을 구성하는 각 프로세스 엔트리의 기본 정보로는 부작업 내에서의 순위(rank), 프로세스 고유번호(pid), 송신자 대기 변수(SW), 수신 대기 플래그(RW)가 요구되며, 각 수신 메시지는 메시지의 길이와 해당 메시지 데이터로 구성하고, 이들은 FIFO 큐로 관리된다.

부작업 내에서 생성된 모든 프로세스는 각각 순위가 부여되고, 해당 순위는 Myjob 서비스 쓰레드가 각 부작업 프로세스를 구분하는데 사용된다. 플래그 RW는 해당 프로세스가 메시지 수신을 위해 대기(blocking) 상태에 있음을 나타내는데 사용되고, 변수 SW는 해당 프로세스로 메시지를 송신하고 대기하는 (확인 응답을 기다리는) 송신 프로세스들의 수를 나타낸다. 이 값은 결국 메시지 큐 내에 저장된 메시지의 수에 해당한다.

메시지 전송방식은 클러스터의 노드 간 통신을 위해서는 UDP(User Datagram Protocol)와 동일 노드 내 통신을 위해서는 System V 지원 IPC인 메시지 큐를 사용하였다. 즉, 프로세스와 통신 지원 서버가 동일 노드에 위치하는 경우에는 UDP가 아닌 메시지 큐를 사용하여 성능 향상을 꾀하였다. UDP는 TCP에 비해 흐름제어, 오류검사, 재전송 등의 기능을 지원하지 않아 상대적으로 안정성이 떨어지나, 클러스터 자원의 네트웍 구성이 비교적 안정되어 있고 작업 동기화에 사용되는 메시지의 크기가 짧고 (4 KB 이내) 단발성이라는 점에서 노드 간



[그림 2] 개선된 자원 관리자의 구성

[표 2] 부작업 내의 프로세스 통신을 위한 API

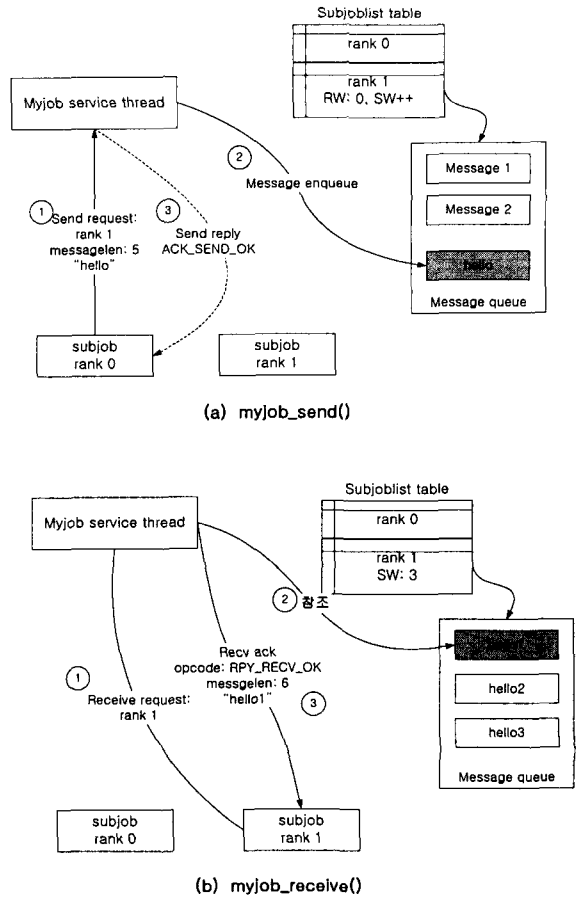
API	기능
myjob_size()	부작업 내 생성된 프로세스의 수를 반환
myjob_rank()	자신의 부작업 내 순위를 반환
myjob_send (rank,message)	rank 프로세스로 message를 송신
myjob_receive()	자신에게 전달된 메시지를 수신
myjob_kill()	부작업 내의 모든 프로세스를 강제 종료

통신 방식으로 UDP를 채택하였다. 이는 TCP가 갖는 세션 수립 오버헤드와 프로세스 이동에 따른 소켓 이동 오버헤드도 성능 저하의 요인으로 판단한데 근거한다.

표 1은 부작업 내의 프로세스 간 통신을 위해 제공되는 myjob_size, myjob_rank, myjob_send, myjob_receive, myjob_kill 등의 API 함수를 나열하고 있고, 각 프로세스는 이를 사용하여 자신의 순위 정보를 얻고, 상대방 프로세스에게 메시지를 전달하거나 자신에게 전달된 메시지를 얻는다.

메시지 송신 API인 myjob_send는 비동기(asynchronous) 방식으로 동작한다. 즉, 그림 3(a)에 보이는 바와 같이, 상대 프로세스(rank)로의 메시지 송신을 Myjob 서비스 쓰레드에게 요청하면, 서버는 메시지를 해당 프로세스 테이블에 저장하고 SW 변수를 증가시킨 후, 바로 응답(Ack) 메시지를 반환함으로써 송신 프로세스를 진행시킨다. 만일 상대방 프로세스가 이미 메시지 수신을 위해 대기 상태에 있으면 메시지 큐로의 입력 과정이 생략되고, 바로 상대에게 전달된다.

이와 달리 myjob_receive는 동기(synchronous) 방식으로 동작한다. Myjob 서비스 쓰레드는 수신 요청한 프로세스를 위한 메시지가 준비되어 있지 않은 경우, RW 플래그를 셋트시켜 해당 프로세스가 수신 대기 중임을 표시하고, 메시지가 도착할 때까지 응답을 지연함으로, 수신 프로세스 대기 상태에 있도록 한다. 그러나 그림 3(b)와 같이 메시지 큐에 이미 메시지가 도착한 경우에는, 최전단의 수신 메시지를 꺼냄과 동시에 메시지의 수를 감소시키고 이 메시지를 해당 프로세스에게 전달한다.



[그림 3] 부작업 프로세스 간 메시지 송수신 과정

4. 결론

클러스터 자원이 그리드 환경에서 효율적으로 운영되기 위해서는 클러스터를 구성하는 노드 간의 부하 균형이 필요하며 이를 위해 실행 중 프로세스 이동은 당연히 지원되어야 한다. 그러나 Globus 툴킷을 비롯한 대부분의 그리드 개발 시스템에서는 작업 시작, 실행 중 동기화 과정 등에서 프로세스들의 실행 중 이동이 고려되지 않았으며, 혹 커널 수준에서 프로세스의 이동을 지원한다 하더라도 TCP 소켓 이전 등의 오버헤드를 일으킨다.

따라서 본 연구에서는 동일 클러스터 내의 프로세스들이 실행 중에 이동하더라도 상호 통신이 가능하도록 별도의 통신 지원 서버 쓰레드를 설계하고, 이를 통해 간접 통신이 이루어지도록 함으로써 각 프로세스의 이동 위치에 무관한 통신 방법을 제안하였다. 또한 클러스터 네트웍이 매우 안정적이고 동기화 과정과 메시지가 단발성인 점에 착안하여 노드 간에는 TCP가 아닌 UDP를, 동일 노드 내 프로세스 간 통신을 위해서는 시스템 V 메시지 큐를 활용하였다.

참고 문헌

[1] I. Foster and C. Kesselman (eds.) "The Grid: Blueprint for a new Computing Infrastructure", Morgan Kaufmann Publishers, 1998.

[2] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", Int'l Journal of Supercomputer Applications, 11(2):115-128, 1997.

[3] K. Czajkowski, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", 2001.

[4] Amnon Barak and Oren La'adan, "The MOSIX *Multicomputer Operating System for High Performance* Cluster Computing", Future Generation

[5] F. Mueller, "Pthreads Library Interface", Technical Report, Florida State University, July 1993.