효율적인 그리드 자원 동시 할당자 구현

김영석[0]* 장재완* 유정록* 함재균** 맹승렬* 이준원*
*한국과학기술원 전자전산학과 전산학전공
**한국과학기술정보연구원 슈퍼컴퓨팅센터
{kimys[0], jwjang, jlyu}@calab.kaist.ac.kr jaehahm@hpcnet.ne.kr {maeng, joon}@cs.kaist.ac.kr

Implementation of Efficient Grid Resource Coallocator

Young-Seok Kim[0]* Jae-Wan Jang* Jung-Lok Yu* Jae-Gyoon Hahm**
Seung-Ryoul Maeng* Joon-Won Lee*
*Div. of Computer Science, Dept. of EECS, KAIST
**KISTI Supercomputing Center

요 약

그리드 시스템이란 네트워크로 연결된 자원들을 통합적으로 관리하여 자원의 효율적인 사용을 지원하기 위한 기반 구조이다. 이러한 환경에서는 응용프로그램의 요구사항을 만족시키기 위해 다중의 자원들을 동시에 할당할 필요성이 자주 발생한다. 본 논문에서는 효율적인 그리드 자원 동시 할당자인 KGB(KAIST GLOBUS) DUROC(Dynamically Updated Resource Online Coallocator)을 구현하였다. KGB *DUROC*은 그리드 미들웨어의 실질적인 표준인 GLOBUS의 DUROC을 GLOBUS IO 라이브러리를 사용하여 소스 코드(source code)수준에서의 호환성을 유지하면서 수정, 개선한 것이다. KGB DUROC 과 GLOBUS DUROC 각각에 마이크로벤치마크를 수행시켜 성능을 분석, 평가하였다.

1. 서 론

그리드 시스템이란 네트워크로 연결된 자원들을 통합적으로 관리하여 자원의 효율적인 사용을 지원하기 위한 기반 구조이다. 즉, 범세계적인 자원 풀을 형성하고, 사용자에게 자원을 적절히 할당하여 사용자 작업을 효과적으로 수행하기 위한 구조와 방법론을 총칭한다 [1]. 그리드 시스템은 기존의 분산 수퍼컴퓨팅 환경에서 수행되던 고성능을 요구하는 복잡한 응용프로그램들의 구축 및 수행에 사용된다. 이러한 응용프로그램들의 공통된 특징은 다중(multiple)의 자원들을 동시에 할당할 필요성이 있다는 것이다 [3]. 이기종의(heterogeneous) 분산된 자원들을 파악하여 사용자에게 동시 할당하는 자원관리(resource management)는 그리드 시스템의 가장 핵심적인 부분이다. 그리드 시스템에서는 이기종의 분산된 자원들을 사용하게 되므로 사용자 인증 및 권한부여, 자원의 발견, 접근 및 할당과 같은 문제들이 발생한다.

그리드 미들웨어는 이러한 문제들을 해결하고 사용자에게 통일된 인터페이스를 제공하는 그리드 시스템의 핵심 기술이다. *GLOBUS*는 그리드 미들웨어의 사실상 표준으로서 여러 계층적인 모듈들로 구성된다. 각 모듈들은 인터페이스를 정의하고, 고 수준의 서비스들은 이 인터페이스를 통해서 각 모듈의 기능을 사용한다 [2]. *GLOBUS*의 자원파악 및 할당 모듈은 *GRAM* (Globus Resource Allocation Manager)과 DUROC으로 구성된다. GRAM은 지역 자원 관리자로서 자신의 영역에 있는 자원들을 관리하고, 그 정보를 MDS(Metacomputing Directory Service) [6] 에게 등록한다. DUROC은 자원 동시 할당자로서 MDS로부터 자원정보를 파악하여 사용자에게 이기종의 분산된 자원을 동시에 할당한다. 이를 위해 *DUROC*과 *GRAM*은 작업 요청, 취소, 수정, 작업상태 확인 및 작업들 간의 통신 인터페이스를 제공한다 [3, 4].

*DUROC*은 *GLOBUS*의 *GRAM*과 통신 모듈인 *NEXUS*와 DUCT를 사용하여 구현되어 있다. 본 논문에서는 NEXUS와 DUCT 모듈의 구조적, 성능적 문제점을 해결하기 위해 GLOBUS DUROC을 수정, 개선하여 효율적인 그리드 자원 동시 할당자인 KGB *DUROC*을 구현하고 성능을 분석, 평가하였다.

본 논문의 구성은 다음과 같다. 2장에서는 GLOBUS DUROC의 구조와 문제점을 설명한다. 3장에서는 이를 해결한 KGB DUROC의 구현에 대하여 기술하며, 4장에서는 구현된 KGB DUROC의 성능을 평가하였고, 5장에서는 결론을 맺는다.

2. *GLOBUS DUROC*

그림1에서 보듯이, 그리드 미들웨어의 사실상 표준인 GLOBUS는 여러 계층적인 모듈들로 구성된다. 그림1의 컨트롤(duroc_control)과 런타임(duroc_runtime) 모듈이 DUROC을 구성한다. 컨트롤 라이브러리는 다중 자원의 동시할당을 요구하는 사용자 작업(job)을 그리드 시스템에서 수행하기 위해, 작업 제출, 취소, 수정, 종료 등의 기능을 제공한다. 런타임 라이브러리는 컨트롤 라이브러리와 함께 제출한 작업에 대한 작업시작(job-start-atomicity) 보장 기능 및 서브작업들간의 통신기능을 제공한다 [7].

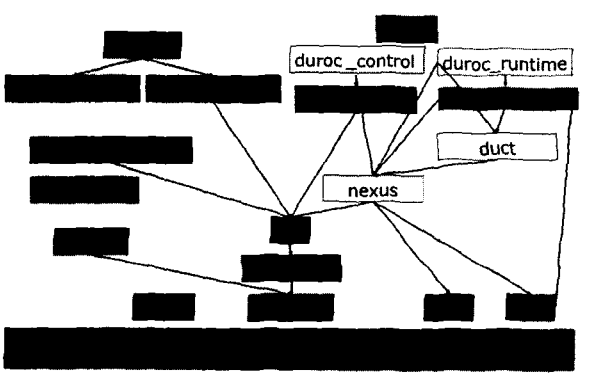

그림1 GLOBUS 모듈의 의존성

그림1에서 보듯이, GLOBUS DUROC의 가장 큰 문제점은 계층을 이루면서 의존성을 가지는 NEXUS와 DUCT를 통신 라이브러리로 사용하는 것이다. NEXUS는 RPC(Remote Procedure Call)형태의 통신 메커니즘을 사용하는 다중쓰래드 통신 라이브러리로서 통신에 필요한 API(Application Programming Interface)를 제공한다 [5]. DUCT는 NEXUS를 토대로 구현한 라이브러리로서 사용자에게 좀더 편리한 인터페이스를 제공한다. NEXUS는 통신을 위해 메시지 전달 방식(message passing)을 사용하는 대신 RPC 스타일의 통신 방식을 사용하기 때문에 양방향 통신이 요구되는 DUROC에는 비효율적이다. 또한 NEXUS는 잠금장치(lock)와 조건변수(condition variable) 같은 동기화 원시함수(synchronization primitive)를 사용하기 때문에 통신 성능이 떨어진다. 결국 GLOBUS DUROC은 NEXUS와 DUCT 라이브러리 계층을 거치면서 실질적인 기능을 수행하기 때문에 성능상의 오버헤드가 크다. 본 논문에서는 이러한 구조적, 성능적 문제점을 개선하여 효율적인 그리드 자원 동시 할당자인 KGB DUROC을 구현 하였다.

3. KGB DUROC

KGB DUROC은 NEXUS와 DUCT 라이브러리에 대한 의존성을 제거하기 위해 GLOBUS IO 라이브러리를 직접적으로 사용하여 GLOBUS DUROC과 같은 기능을 제공하도록 구현하였다. GLOBUS IO 라이브러리에서 제공하는 API는 TCP/IP 소켓(socket) API의 래퍼(wrapper) 함수들이다 [7].

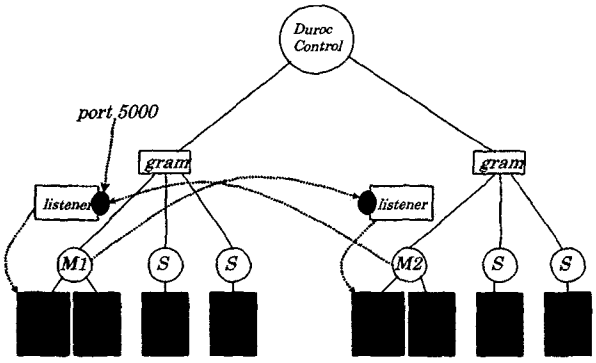

그림2 KGB DUROC 동작 메커니즘

KGB DUROC은 그림2에서 설명하는 메커니즘으로 동작한다. 그리드 시스템에서 사용자 작업(job)은 하나 이상의 서브작업(subjob)들로 구성되고, 각 서브작업은 하나 이상의 프로세스들로 구성된다. 서브작업을 구성하는 프로세스들은 계급(rank)을 가진다. 계급이 0인 프로세스를 마스터프로세스(master process)라고 부르고, 그 외의 프로세스를 슬래이브프로세스(slave process)라고 부른다. 또한 마스터프로세스들 간의 통신을 인터서브작업(inter-subjob) 통신이라 하고, 그 외의 프로세스들 간의 통신을 인트라서브작업(intra-subjob) 통신이라 한다. 사용자는 DUROC 컨트롤 라이브러리에서 제공하는 API를 사용하여 작업(job)을 제출한다. 제출한 작업은 서브작업들로 나누어져 여러 사이트(site)에 분배되고, 분배된 서브작업은 사이트에서 하나이상의 프로세스로 나누어져 실행에 필요한 자원을 얻는다. 프로세스들이 자원을 얻어 실행준비가 완료되면 작업시작을 보장하기 위해 런타임에서 제공하는 API를 이용하여 체크인 프로토콜에 따라 체크인을 하고, 컨트롤의 배리어해제(barrier release)를 기다린다 [7]. 배리어해제가 되면 마스터프로세스는 런타임에서 제공하는 인트라서브작업 통신 API를 사용하여 수행에 관한 명령을 슬래이브프로세스들에게 전달하고 자신도 그 명령에 따라 작업을 수행한다. 작업 수행중인 프로세스는 서로 다른 GRAM 하에 수행중인 프로세스와 통신하기 위해 인터서브작업 통신 API를 사용하여 메시지를 주고받는다. 그림2에서 마스터프로세스 M1이 마스터프로세스 M2와 통신하기 위해 5000번의 잘 알려진(well-known) 포트로 메시지를 보낸다. 이때 M2의 리스너(listener)는 5000번 포트에 메시지가 도착하면 그 메시지를 인터테이블(inter table)에 저장한다. 최종적으로 M2는 인터테이블을 검색하여 M1의 메시지를 찾아간다.

KGB DUROC의 구현에 있어 크게 2가지 사항을 고려하였다. 첫째 기존의 GLOBUS를 기반으로 개발된 프로그램과의 호환성 유지, 둘째 변경된 통신 라이브러리 사용에 따른 효율적인 통신 프로토콜 개발 이다. GLOBUS를 기반으로 개발된 프로그램과의 호환성 유지를 위해 GLOBUS DUROC의 API와 완벽한 호환성을 제공한다. NEXUS와 DUCT 라이브러리를 사용하지 않고 GLOBUS IO 라이브러리를 직접적으로 사용하기 때문에 실행파일(binary executable) 수준의 호환성은 잃어버리게 되지만, 소스코드(source code) 수준의 호환성은 완벽히 유지하였다. KGB DUROC에서는 통신 라이브러리인 NEXUS와 DUCT의 사용을 배제하였기 때문에 기존의 통신 프로토콜을 사용할 수 없게 되었다. 기존의 통신 프로토콜을 대체하기 위해 체크인 프로토콜, 배리어해제(barrier release) 프로토콜, 작업들간 구조(inter subjob structure) 프로토콜과 같은 3개의 프로토콜을 GLOBUS IO 라이브러리를 사용하여 구현하였다.

4. 성능 평가

본 논문에서 구현한 KGB DUROC 라이브러리의 성능평가를 위해 마이크로벤치마크를 KGB DUROC과 GLOBUS DUROC에 대해 각각 수행 시킨 후 그 결과를 분석하였다. 다음과 같이 크게 두개의 범주로 나누어서 성능평가를 하였다. 먼저 라이브러리 초기화와 배리어해제(barrier release) 시점까지의 지연시간을 측정하였고, 다음으로 서브작업(subjob)들 간의 통신 성능을 측정하기 위해 양방향 지연시간 및 전송 대역폭을 측정하였다.

실험을 위해서 Pentium-III 700MHz CPU, 512MB 메모리, BCM5700 Gigabit Ethernet NIC 그리고 Linux version 2.4.18 이 운영체제로 설치된 PC를 2대 사용하였다.

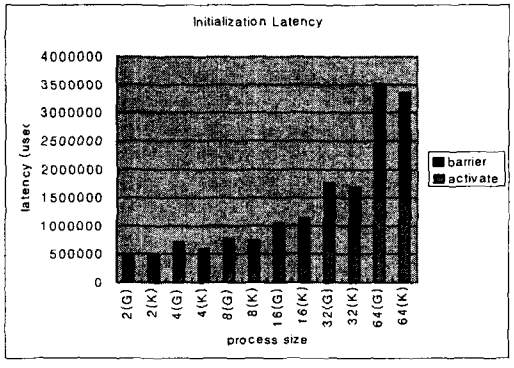

그림3 초기화 지연시간

그림3은 서브작업에 속하는 프로세스의 수를 2개에서부터 64개까지 증가시키면서 라이브러리의 초기화와 배리어해제 시점

까지의 누적된 지연시간을 보여준다. X축의 괄호 안에 G로 표시된 것은 GLOBUS DUROC을 나타내고 K로 표시된 것은 KGB DUROC을 나타낸다. 초기화 지연시간을 비교해 볼 때 GLOBUS와 KGB는 거의 같은 성능을 보이고 있다. 배리어해제 지연시간을 비교해 보면 KGB가 GLOBUS에 비해 평균적으로 약 5%정도의 향상된 성능을 보이고 있다. 이것은 GLOBUS IO를 사용하여 체크인, 배리어해제 프로토콜을 구현함으로써 기존 통신 프로토콜의 오버헤드를 개선한 결과이다.

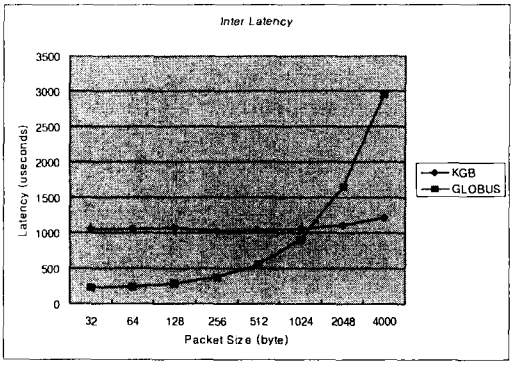

그림4 양방향 지연시간(round trip latency)

그림4는 서브작업들 사이의 양방향 지연시간을 측정한 결과를 보여준다. 이 실험을 위해 두개의 서브작업 사이에 메시지 사이즈를 32바이트에서 4K바이트까지 증가시키면서 그 지연시간을 측정하였다. 1K바이트를 초과하는 메시지에 대해서 KGB DUROC이 훨씬 더 좋은 성능을 보이고 있다. GLOBUS는 메시지 사이즈가 증가함에 따라 지연시간이 급격히 증가하는데 반해 KGB는 메시지 사이즈가 증가하더라도 거의 선형적인 지연시간의 증가만을 보여주고 있다.

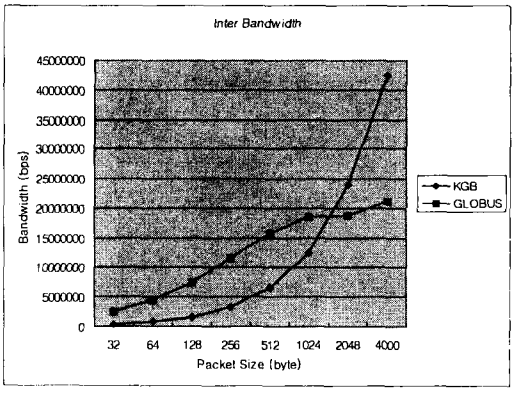

그림5 전송 대역폭(bandwidth)

그림5는 메시지 사이즈를 32바이트에서 4K바이트까지 증가시키면서 서브작업들 사이의 전송 대역폭을 측정한 결과이다. 양방향 지연시간 측정 때와 같이 KGB DUROC은 1K바이트를 초과하는 메시지에 대해 훨씬 더 좋은 성능을 보이고 있다.

DUROC의 서브작업은 하나의 마스터프로세스와 여러 개의 슬래이브프로세스들로 구성되어 있는데, 서브작업들 간의 통신은 마스터프로세스만이 할 수 있고, 프로세스들 간의 통신은 어느 프로세스나 할 수 있다. 이렇게 통신이 가능한 범주가 계층적으로 구분 되어 있다. 따라서 서로 다른 서브작업들에 속하는 슬래이브프로세스들 간의 통신이 필요한 경우, 마스터 노드는 슬래이브프로세스들로부터 받은 메시지를 모아서 다른 서브작업의 마스터프로세스에게 전달하는 경우가 빈번하게 발생하게 된다. 이러한 관점에서 볼 때 큰 메시지 사이즈에 대해 우수한 성능을 보이는 KGB DUROC이 GLOBUS DUROC보다 더 효율적이다. 1K바이트 이하인 메시지를 전송할 경우 KGB가 GLOBUS보다 좋지 못한 성능을 보이고 있는데 이것은 코드상의 최적화를 통해서 성능 향상이 가능할 것으로 보인다.

5. 결론

본 논문에서는 그리드 환경에서 이기종의 분산된 자원들을 동시에 효율적으로 할당하기 위해 그리드 자원 동시 할당자인 KGB DUROC을 구현하고 성능을 평가하였다. KGB DUROC은 GLOBUS DUROC의 변형으로 GLOBUS에서 제공하는 NEXUS와 DUCT 같은 계층적인 모듈 의존성을 제거하기위해 IO 라이브러리를 직접적으로 사용하여 구현한 그리드 자원 동시 할당자이다. KGB DUROC은 GLOBUS에 기반하여 개발된 프로그램을 위해 소스 수준에서의 호환성을 유지한다. 또한 구현에 사용한 라이브러리의 변화에 따라 새로운 통신 프로토콜을 구현하였다. KGB DUROC을 사용하면 GLOBUS DUROC에 비해 초기화 하는데 있어서 평균 5%정도의 성능 향상을 볼 수 있고, 통신 성능에 있어서 1K바이트 이상인 메시지에 대하여 월등한 통신 성능 향상을 기대 할 수 있음을 확인하였다.

참고문헌

[1] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Intl. Journal of Supercomputing Applications*, 2001.

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

[3] K. Czajkowski, I. Foster, and C. Kesselman. Resource Co-Allocation in Computational Grids. *Proc. of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC-8), pp. 219-228*, 1999.

[4] K. Czajkowski, I. Foster, N. Karonis, S. Martin, W. Smith, S. Tuecke. A Resource Management Architecture for Metacomputing Systems. *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pp.62-82*, 1998.

[5] I. Foster, C. Kesselman, S. Tuecke. The Nexus Approach to Integrating Multithreading and Communication. *Journal of Parallel and Distributed Computing, 37:70-82*, 1996.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman. Grid Information Services for Distributed Resource Sharing. *Proc. of Tenth IEEE Intl. Synposium on High-Performance Distributed Computing, IEEE Press*, August 2001.

[7] GLOBUS API reference
http://www.globus.org/developer/api-reference.html