MPI 환경에서 자원 사용량 측정을 위한 어카운팅 시스템 개발

황 호 전[†] · 안 동 언[††] · 정 성 종[†††]

요 약

유닉스 계열의 운영체제에서 사용되는 로컬 어카운팅 시스템은 하나의 호스트상에서 동작하는 프로세스의 어카운팅 정보를 제공한다. 그러나 분산 처리 환경에서 전통적인 로컬 어카운팅 시스템은 동일 작업을 수행하는 프로세스들의 전체 자원 사용량 데이터를 기록하지 못한다. 따라서 본 논문에서는 클러스터 환경에서 MPI(Message Passing Interface) 작업에 대한 자원 사용량 데이터를 측정하고, 관리할 수 있는 어카운팅 시스템을 개발한다. 각 클러스터 노드에 병렬 작업을 수행하는 프로세스의 자원 사용량 데이터와, 병렬 작업을 처리하기 위해 협력하는 프로세스들간의 네트워크 접속 정보를 기록하는 로컬 어카운팅 시스템을 구현한다. 그리고 각 노드의 로컬 어카운팅 시스템에서 기록된 자원 사용량 데이터를 수집하여, MPI 작업 단위의 어카운팅 정보를 만들어 내는 어카운팅 시스템을 개발한다. 마지막으로 대규모 클러스터링 환경에서 널리 사용되는 로컬 스케줄러들에 의해 측정된 자원 사용량 데이터 항목들과 비교 평가한다.

키워드 : 어카운팅 시스템, 자원 사용량, LKM

Development of Accounting System to Measure the Resource Usage for MPI

Ho-Joen Hwang[†] · Dong-Un An[††] · Seung-Jong Chung[†††]

ABSTRACT

Local accounting system used by UNIX-like operating system provides the accounting information of processes that are in single host. But it is impossible for this local accounting system to record the total resource consumption data of all processes for doing the same job simultaneously. In this paper, we implement accounting system to measure and manage resource usage for MPI(Message Passing Interface) job in the clustering environment. We designed and implemented the accounting system which measure resource usage of each process runs on a cluster node and record the interconnection information of the entire set of processes across network. Also we implemented accounting system which collect the resource usage data of process in the local accounting system and generate the job-level accounting information. Finally, to evaluate the resource consumption data measured by this accounting system, we compare with the data collected by local scheduler that widely used in large scale clustering environment.

Key Words : Process Accounting, Resource Usage, LKM

1. 서 론

클러스터링 시스템은 네트워크로 연결되어 있는 각각의 컴퓨터들이 서로 협력하여 각 시스템의 CPU, 메모리, 저장장치등을 공유하여 사용자 및 응용 프로그램에 단일 이미지를 제공하는 시스템을 의미한다[1]. 특히 PVM(Parallel Virtual Machine), MPI(Message Passing Interface)와 같은 병렬 처리 라이브러리를 지원하는 클러스터링 시스템은 하나의 큰 작업을 여러 클러스터 노드에 분산 처리시켜, 노드들간의 정보 교환을 통해 거대하고 복잡한 문제를 해결하도록 한다.

이와같이 PVM이나 MPI와 같은 병렬 처리 라이브러리를 기반으로 하고 있는 클러스터링 시스템은 마스터 노도(Master Node)와 슬레이브 노드(Slave Node)로 구성되어 있다. 마스터 노드는 각 슬레이브 노드에 작업 할당 및 작업 전체에 대한 총괄적인 제어와 관리를 책임지고, 슬레이브 노드는 마스터 노드로부터 할당받은 작업을 실질적으로 처리한다. 거대하고 복잡한 병렬 작업은 각 노드에서 수행될 Task 단위로 분할되어 여러 노드에서 동시에 처리된다. 또한 프로

† 준 회 원 : 전북대학교 컴퓨터공학과 박사과정
†† 종신회원 : 전북대학교 컴퓨터공학과 부교수
††† 정 회 원 : 전북대학교 컴퓨터공학과 교수
논문접수 : 2004년 12월 13일, 심사완료 : 2005년 4월 15일

세스들간의 메시지 교환을 통해 서로 협력하여 병렬 작업을 처리한다. 그러므로써 작업 처리량을 증대시킬 수 있을 뿐만 아니라 주어진 시간내에 복잡한 문제를 빠르게 해결할 수 있도록 해준다.

자원 제공자는 다양한 컴퓨팅 자원들을 공유함으로써 수많은 사용자들의 복잡한 문제를 신속하게 해결할 수 있도록 해준다. 그리고 자원 제공자는 사용자의 작업을 처리해주는 대가로 자원 사용에 대한 요금을 부과할 수 있다. 다시 말해, 각 클러스터 노드에서 사용자의 작업을 처리하는데, 어떠한 자원들을 얼마만큼 소비했는지를 측정하여 사용료를 부과할 수 있다. 따라서 이런 클러스터링 환경에서는 작업 어카운팅 시스템이 반드시 필요하게 된다[2][3].

작업 어카운팅 시스템은 각 노드에서 병렬 또는 단일 작업을 처리하는 태스크 프로세스들이 소비한 자원 사용량 데이터를 측정하고, 흩어져 있는 데이터들을 수집하여 작업별로 통합하고 분석한다[2][4]. 또한 작업 어카운팅 데이터는 요금 부과 알고리즘을 거쳐 사용자에게 과금 서비스가 가능하도록 한다.

따라서 클러스터링 환경에서 작업 어카운팅 시스템을 지원하기 위해선, 맨 먼저 사용자의 작업을 처리하는데 소비한 자원 사용량을 정확하게 측정하는 것이 아주 중요하다. 이를 위해 각 노드에서 프로세스들의 어카운팅 데이터를 작성할 수 있는 로컬 어카운팅 시스템이 필요하고, 각 노드의 로컬 어카운팅 시스템으로부터 수집된 데이터들을 작업 단위의 어카운팅 데이터로 취합할 수 있는 Accumulation Routine이 필요하다.

현재 대규모의 클러스터링 시스템에서 사용하고 있는 LoadLeveler, PBS(Portable Batch System), LSF(Load Sharing Facility)와 같은 로컬 스케줄러들은 사용자 병렬 작업에 대해서 어카운팅 정보를 제공해주고 있다[5][6]. 그러나 운영체제에서 제공하는 전통적인 로컬 어카운팅 시스템을 사용하는 클러스터링 환경에서는 그렇지 못하다. Fork 로컬 스케줄러로 MPI 병렬 작업을 처리하는 경우, 각 노드에 설치된 로컬 어카운팅 시스템으로부터 태스크 프로세스 단위의 어카운팅 데이터가 측정되지만, 동일 병렬 작업을 수행하는 프로세스들간의 어카운팅 정보를 취합하는 어큐뮬레이션 루틴을 작성할 수 없다.

왜냐하면 각 노드에 설치된 로컬 어카운팅 시스템은 단일 호스트 상에서 수행되는 프로세스의 자원 사용량 데이터만을 기록할 뿐, 동일 작업에 대해 다른 호스트 상에서 수행하는 프로세스들에 대한 상호 관련 정보는 기록하지 않는다. 따라서 여러 클러스터 노드에서 동일 병렬 작업을 수행하는 모든 프로세스들의 총 자원 사용량 데이터를 취합할 수 없다.

그러므로, 본 논문은 MPI 병렬 작업의 프로세스 자원 사용량을 측정하는 어카운팅 시스템을 개발하기 위해, 각 클러스터 노드에 병렬 작업을 수행하는 프로세스의 자원 사용량 데이터와 상호 협력하는 프로세스들간의 네트워크 접속 정보를 작성할 수 있는 로컬 어카운팅 시스템을 설계한다. 또한 각 노드의 로컬 어카운팅 시스템으로부터 측정된 데이터를 수집, 분석한 다음, 작업 단위의 어카운팅 정보를 생산하는 시스템을 개발한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대하여 살펴보고, 3장에서는 MPI 라이브러리 기반 클러스터링 시스템에서 병렬 작업의 처리과정을 기술한다. 4장에서는 논 본문에서 개발한 MPI 작업 자원 사용량 측정을 위한 어카운팅 시스템을 설명하고, 5장에서는 실험 환경 및 실험 결과, 마지막으로 6장에서 결론을 맺는다.

2. 관련 연구

네트워크 상에서 분산되어 있는 이기종 컴퓨팅 자원, 프로세싱 그리고 데이터 및 대용량 스토리지 시스템들을 통합하고 연결하는 Grid[7] 컴퓨팅 기술이 국내외적으로 활발한 연구가 진행되고 있다. 그리드 컴퓨팅 기술은 인터넷상에 분산되어 있는 미사용 자원을 활용하여 방대한 양의 데이터와 컴퓨팅 성능을 요구하는 복잡한 문제 해결하기 위해 사용된다.

그리드 환경에서 자원 제공자는 자원 공유 및 유휴자원 활용률을 향상시키기 위해 컴퓨팅 자원을 제공하고, 자원 소비자는 단일 시스템으로 해결할 수 없는 작업들에 대해 그리드 자원을 활용함으로써 이를 해결할 수 있다. 이와 같이 LoadLeveler, PBS, LSF, Fork등 다양한 로컬 스케줄러를 사용하는 클러스터링 시스템들이 그리드 컴퓨팅 환경에서 어카운팅 서비스를 하기 위해 많은 연구가 진행 중이다. 그리드 어카운팅 시스템은 그리드 자원들에 대한 인증 기능, 각 자원에 대한 접근 및 사용 권한 기능, 사용자의 작업을 해결하기 위해 사용된 자원에 대한 비용 산출 서비스 기능으로 구성된다.

그리드 어카운팅 관련 시스템으로는 DataGrid 프로젝트에서 개발된 DGAS(DataGrid Accounting System)[8], GridX (Grid Exchange) Meta-Scheduler Development[3] 프로젝트의 GridMonitor 모듈이 있다. 특히 GridX Meta-Scheduler Development 프로젝트에서 그리드 어카운팅 구조는 GGF (Global Grid Forum)의 OGSA-WG(Open Grid Service Architecture Working Group)에서 제안한 구조[2]를 따르고 있으며, 자원 사용량 측정 데이터 항목으로는 GGF(Global Grid Forum)의 UR-WG(Usage Record Working Group)에 제안한 데이터들을 측정하였다. URWG에서는 여러 사이트들 간에 어카운팅 데이터들을 교환하기 위한 공통된 포맷을 정의하였다[4].

GGF의 RUS-WG(Resource Usage Service Working Group)

의 GSAX(Grid Service Accounting Extensions)에서 그리드 어카운팅 시스템 프레임워크를 제안하였다[2]. 이 제안된 프레임워크는 자원 사용량을 수집하는 Monitoring, 요금 정책을 적용하는 Metering, 자원 제공자와 자원 소비자간의 사업적 관계를 관리하는 Accounting으로 구성되어 있다. 이 구성요소들중 Monitoring 구성요소에서는 그리드 서비스에 대한 자원 사용량을 측정할 수 있는 아키텍처를 제안하고 있다.

그리드 컴퓨팅 환경을 구축하기 위해 Globus Toolkit[9][10] 미들웨어가 널리 사용된다. 특히 GRAM(Globus Resource Allocation Management)은 원격지의 자원들을 동시에 사용할 수 있도록 해주고, 관리하는 글로버스 툴킷의 핵심적인 요소이다. 따라서 GRAM은 Condor, EASY, Fork, Load Leveler, LSF, NQE(Network Queuing Environment)와 같은 로컬 스케줄러와의 인터페이스를 가지고 있다[11]. 따라서 그리드를 위한 어카운팅 시스템을 개발하기 위해서 각 로컬 스케줄러들은 그리드 사용자 작업을 처리하는데 소비한 자원 사용량을 측정할 수 있어야 한다.

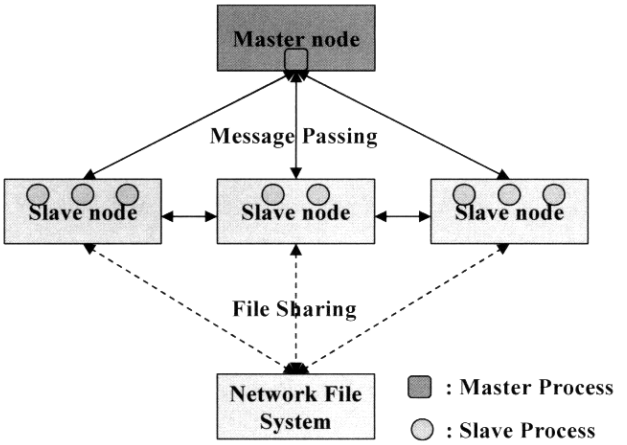
대용량 클러스터링 환경에서 널리 사용되고 있는 Load Leveler, LSF, PBS와 같은 로컬 스케줄러들은 단일 및 병렬 작업에 대한 프로세스 어카운팅 데이터를 제공해주지만, 유닉스 계열의 운영체제에서 사용하는 Fork 로컬 스케줄러는 병렬 작업에 대한 프로세스 어카운팅 정보를 제공하지 못한다. 따라서 Fork 로컬 스케줄러를 사용하고 있는 클러스터링 시스템도 그리드 어카운팅 서비스를 제공하기 위해서는 MPI 병렬 작업에 대한 자원 사용량을 측정하고, 관리할 수 있는 어카운팅 시스템이 반드시 필요하다.

3. 병렬 처리 라이브러리 기반 클러스터링 시스템

3.1 MPI(Message Passing Interface) Model

대부분의 클러스터링 시스템은 분산된 컴퓨팅 자원(프로세서와 메모리등)들간의 통신이 가능하도록 하나의 고속 네트워크로 연결된 구조이다. 개념적으로 볼 때, 고유의 계산 능력을 가진 각 클러스터 노드는 하나의 네트워크로 연결된 모든 컴퓨팅 자원들과 통신할 수 있는 독립적인 프로세서라고 볼 수 있다. 따라서 이러한 컴퓨팅 환경에 밀접한 관련이 있는 프로그래밍 모델이 Message Passing Interface Model이다. 메시지 패싱 인터페이스 모델[12]은 각각의 프로세스가 독자적인 메모리 공간을 가지고 있어 각 클러스터 노드의 프로세스들간의 메시지를 교환함으로써 하나의 문제를 해결해 나가는 프로그래밍 모델이다(그림 1).

따라서 오늘날 MPI 프로그래밍 모델이 분산 환경에서 일반적으로 널리 사용된다. 국제 표준으로 채택된 MPI는 병렬/분산 환경에서 TCP 소켓과 UNIX의 RSH(Remote SHell)을 이용하여 서로 다른 클러스터 노드들간의 통신을 할 수



(그림 1) Message Passing Interface Model

있도록 해 주는 라이브러리이다. 그러므로 MPI 라이브러리는 이기종 자원들간의 정보 교환을 지원하는 이질성, 컴퓨팅 자원을 쉽게 추가 확장할 수 있는 확장성, 그리고 서로 다른 컴퓨팅 자원에서도 실행시킬 수 있는 이식성등 다양한 장점을 지니고 있다.

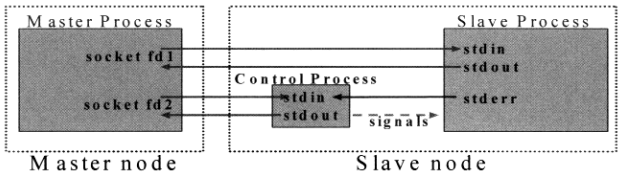
3.2 MPI 기반 병렬 작업 처리

병렬 처리 라이브러리 기반 클러스터링 환경에서, 마스터 노드에서는 사용자의 MPI 작업을 분산 처리하기 위해 UNIX의 RSH 명령어를 통해 각 슬레이브 노드들에게 작업 처리 요청을 한다. RSH 명령어는 지역 호스트의 프로그램을 다른 원격 호스트에서 처리할 수 있도록 해 주는 원격 명령어 실행이다. 마찬가지로 MPI 작업도 각 클라이언트 노드에서 처리될 태스크로 분할 한 다음, RSH 명령어를 통해 분할된 태스크들을 해당 노드에서 처리 하도록 한다.

마스터 노드에서 MPI 작업 전체의 총괄적인 제어를 담당하는 프로세스를 마스터 프로세스라고 하고, 각 슬레이브 노드에서 실질적인 작업을 처리하는 프로세스를 슬레이브 프로세스라고 한다(그림 1).

(그림 2)는 원격 명령어를 처리하는 마스터 프로세스와 슬레이브 프로세스간의 데이터 전송 및 실행 제어 메커니즘을 보여준다. 마스터 프로세스와 슬레이브 프로세스 사이에 서로 다른 두개의 TCP 소켓을 사용하여 슬레이브 프로세스의 표준 출력과 표준 에러를 구분해 준다. 소켓 기술자 socket fd1을 통해 마스터 프로세스는 슬레이브 프로세스의 표준 입력 데이터를 전송하고, 그 반대로 슬레이브 프로세스의 표준 출력을 전송받는다. 그리고 마스터 프로세스는 소켓 기술자 socket fd2를 통해서 슬레이브 프로세스의 표준 에러를 읽을 수 있으며, 또한 이 소켓에 시그널을 전송하여 슬레이브 프로세스의 실행을 제어한다[13].

MPI 라이브러리를 사용하는 병렬 작업은 슬레이브 노드에서 처리될 태스크 단위로 분할되어, 태스크 할당 정책에



(그림 2) 프로세스들간의 데이터 전송 및 실행 제어

따라 각 슬레이브 노드로 전달된다. 이때 동일 병렬 작업을 수행하는 태스크 프로세스들의 집합을 MPI Communicator라고 한다. MPI Communicator내에 각각의 태스크 프로세스들을 식별할 수 있는 Rank(Process's Rank)를 부여하고, 이 Rank를 통해 해당 프로세스들간의 Point-to-Point 통신 방법으로 메시지를 주고 받을 수 있다.

4. MPI 환경에서 자원 사용량 측정을 위한 어카운팅 시스템 설계 및 구현

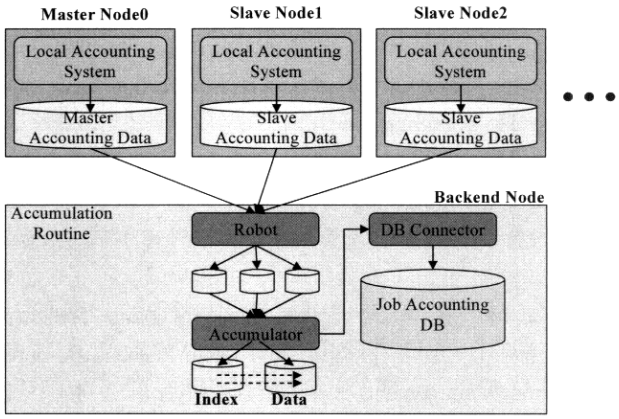
MPI 환경에서 병렬 작업 단위의 어카운팅 정보를 측정하기 위해 마스터 노드와 슬레이브 노드에 로컬 어카운팅 시스템을 배치하였다. 그리고 백엔드(Backend) 노드에 어큐뮬레이션 루틴을 두어 마스터 노드와 슬레이브 노드에서 측정된 자원 사용량 데이터를 취합할 수 있도록 하였다(그림 3).

MPI 병렬 작업은 마스터 노드에서 mpirun 명령어를 통해 각 슬레이브 노드에서 처리될 태스크로 분할하여 전달한다. 이 때, MPI 병렬 작업을 구동시키는 *mpirun* 명령어는 기본적으로 관리자의 할당 정책 파일(machines.LINUX)을 참고하여, 각 슬레이브 노드로 태스크들을 분배한다. 또한 사용자가 정의한 할당 정책이 반영되도록 하기 위해서는 -machinefile이란 옵션을 이용한다.

(그림 3)에서, 각 노드에 설치된 로컬 어카운팅 시스템은 로컬 머신 상에서 실행하는 모든 프로세스들을 감시하여 MPI 작업을 수행하는 프로세스들을 찾아내, 이들 프로세스들의 자원 사용량을 측정하고 지정된 어카운팅 파일에 저장하였다. 그리고 백엔드 노드에 설치된 어큐뮬레이션 루틴을 통해 각 노드에 저장된 어카운팅 파일들을 주기적으로 수집하여, 동일 병렬 작업을 수행한 태스크 어카운팅 데이터들을 취합한 후에 작업 단위에 어카운팅 정보를 생산하였다.

각 노드에 설치된 로컬 어카운팅 시스템에서는 모든 클러스터 노드의 프로세스 스케줄링 정책을 보장하고, MPI 병렬 작업의 어카운팅 정보를 측정하기 위해 생성되는 모든 프로세스들을 실시간으로 모니터링 하였다. 그리고 각 로컬 어카운팅 시스템은 MPI 병렬 작업을 처리하는 프로세스의 활동을 추적하여 MPI Communicator에 참여하는 프로세스들과의 TCP 소켓 정보를 찾아내고, 해당 프로세스가 종료될 때 자원 사용량 데이터와 소켓 정보를 기록하였다.

따라서, MPI 작업 어카운팅 데이터는 MPI Communicator내의 모든 태스크 프로세스들의 자원 사용량 정보와 상호 협력하는 프로세스들간의 네트워크 접속 정보로 구성하였다. 네트워크 접속 정보는 마스터 프로세스와 슬레이브 프로세스들이 동일 MPI Communicator에 참여한 프로세스들임을 식별하기 위한 TCP 소켓 정보를 의미한다.



(그림 3) MPI 작업 자원 사용량 측정을 위한 어카운팅 시스템

4.1 프로세스 모니터링

모든 클러스터 노드에서 로컬 어카운팅 데이터를 수집하기 위해서는 사용자의 병렬 작업을 수행하는 태스크 프로세스들의 생성 및 활동을 실시간으로 모니터링할 수 있어야 한다. 유닉스 계열의 운영체제에 포함되어 있는 디폴트 감사 시스템(Syslog, Klog)은 커널 공간상에서 생성된 로그나 여러 데몬들이 생성하는 로깅 관련 데이터들만을 처리하고 있다.[14] 운영 체제의 디폴트 감사 시스템에서 제공하는 정보만을 가지고는 호스트상에 존재하는 모든 프로세스들의 활동을 감시하기에는 역부족하다.

그래서 각 노드의 로컬 어카운팅 시스템은 모든 프로세스들의 활동을 실시간으로 모니터링하기 위해, 커널 공간상에서 프로세스들을 감시할 수 있는 LKM(Loadable Kernel Module)을 추가하여 구현하였다. LKM은 커널 공간상에 적재 가능한 커널 모듈로, 운영체제에서 제공되지 않는 기능들에 대해서 추가/삭제가 가능한 커널의 구성요소이다. LKM 기법을 이용한 확장 모듈을 사용해 호스트상의 모든 프로세스들의 활동을 모니터링 할 수 있게 된다.

U.C. Davis의 Linux BSM[15]과 SNARE[16]는 LKM 기법을 활용한 시스템 콜 래핑(System Call Wrapping)을 통해 감사 자료를 생성해주고 있다. 이와 같이 커널 모듈 확장을 통해 사용자가 원하는 기능을 커널 공간상에 추가시킬 수 있으며, 그 기능이 필요할 때마다 동적으로 로딩, 언로딩이 가능하다. 또한 커널 모듈을 변경하더라도 커널 재컴파일 및 재부팅 과정이 필요없어 프로세스 감사 시스템에 널리 활용되는 기법이다.

<표 1>은 시스템 콜 래퍼 모듈들이 감시해야할 프로세스

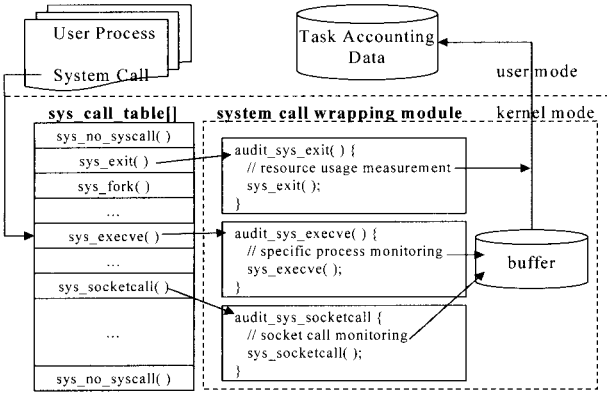
〈표 1〉 사용자 환경 설정 파일

```
# Job Accounting Information Configuration file

# MPI Job Task Accounting Log file
log             /var/log/pacct

# Remote Shell Daemon file
shell           /usr/sbin/in.rshd

# MPI Run file
mpirun          /usr/local/mpich/bin/mpirun
```



(그림 4) 프로세스 모니터링 및 로컬 어카운팅 시스템

들의 정보와 로컬 어카운팅 데이터를 기록할 로그파일을 명시하는 사용자 환경 설정 파일이다. 이 설정 파일 정보는 커널 모듈이 적재될 때 메모리 기억공간에 저장되고, 시스템 콜 랩퍼들에 의해서 참조된다.

MPI 병렬 작업을 수행하는 프로세스들을 찾아내기 위해 sys_execve 시스템 콜을 가로채 시스템상에 실행되는 모든 프로세스들을 실시간으로 모니터링 하였다. 프로그램을 실행시키기 위해서는 커널 공간상에서 sys_execve 시스템 콜을 호출하기 때문이다. 따라서 sys_execve 시스템 콜을 가로채 실행 파일들의 이름을 검사하여 병렬 작업을 수행하는 프로세스들을 탐지하였다(그림 4).

마스터 노드에서는 병렬 작업을 구동시키는 mpirun 프로세스와 이 프로세스로부터 파생된 프로세스들 중에 병렬 작업을 처리하는 마스터 프로세스를 찾아 활동을 감시하였다. 그리고 슬레이브 노드에서는 원격 명령어를 실행하는 프로세스를 찾고, 이 프로세스로부터 파생된 슬레이브 프로세스를 감시하였다. 그리고 슬레이브 노드에서는 여러개의 슬레이브 프로세스들이 동시에 수행될 수 있기 때문에 각각의 태스크 프로세스들은 세션 아이디로 구분하였다.

4.2 프로세스 네트워크 접속 정보 수집

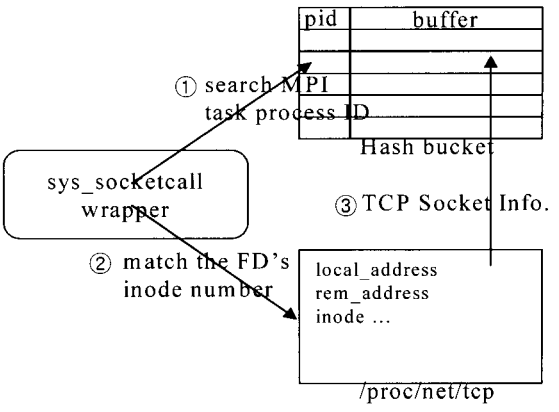
병렬 작업 어카운팅 데이터를 측정하기 위해, 각 클러스터 노드에서 협력하는 프로세스들간의 소켓 정보를 찾아내는 것이 아주 중요하다. 왜냐하면 동일 병렬 작업을 수행하는 프로세스들간의 상호 연결된 네트워크 접속 정보를 통해 MPI Communicator내에 참여하는 프로세스의 집합을 찾아낼 수 있기 때문이다. 따라서 각 클러스터 노드에서는 sys_socketcall 시스템 콜을 가로채, MPI 작업을 수행하는 프로세스의 네트워크 접속 정보를 찾아낼 수 있도록 처리하였다.

MPI 병렬 작업을 참여하는 프로세스들간의 네트워크 접속 정보를 찾아내기 위해 마스터 프로세스에서는 모든 슬레이브 프로세스들과의 TCP 소켓 정보를 찾고, 각 슬레이브 프로세스들은 자신과 입출력하고 있는 마스터 프로세스와의 TCP 소켓 정보를 찾는다. 이와 같이 해당 프로세스의 TCP 소켓 정보(Local IP and Port, Remote IP and Port)를 얻기 위해 소켓 기술자를 추적해서 아이노드 구조체의 아이노드 번호와 /proc/net/tcp 파일을 참고하여 네트워크 접속 정보를 얻어냈다(그림 5).

PROC 파일 시스템은 유닉스 계열의 운영체제에서 커널 공간상의 프로세스 활동 상황 및 통계 정보, CPU 사용률, 네트워크 접속 상태, 적재된 모듈 등 광범위한 정보를 사용자 공간상에서도 쉽게 접근할 수 있도록 설계된 가상 특수 파일 시스템이다. 그리고 /proc/net/tcp 파일은 TCP 소켓 테이블을 덤프한 내용을 담고 있어, 이 파일로부터 마스터 프로세스와 슬레이브 프로세스간의 연결된 소켓의 로컬 주소와 포트 번호 그리고 원격지 주소와 포트 번호를 얻어낸다. 또한 커널 공간상에서 TCP 소켓이 어느 아이노드 객체를 통해서 입출력이 이루어지는지를 식별할 수 있는 아이노드 객체 번호도 포함되어 있다.

MPI 병렬 작업은 마스터 노드에서 태스크 단위로 분할하여 각 슬레이브 노드에 태스크들을 할당한다. 그리고 각 노드의 슬레이브 프로세스들은 마스터 프로세스로부터 넘겨받은 IP 주소와 포트 번호를 통해 상호 정보 교환이 이루어진다.

각 슬레이브 프로세스들은 전달된 IP 주소와 포트번호로 마스터 프로세스에 소켓 접속을 요청한다. 따라서 마스터 프로세스는 각 슬레이브 프로세스들의 소켓 접속 요청을 허



(그림 5) 프로세스의 TCP 접속 정보 검색

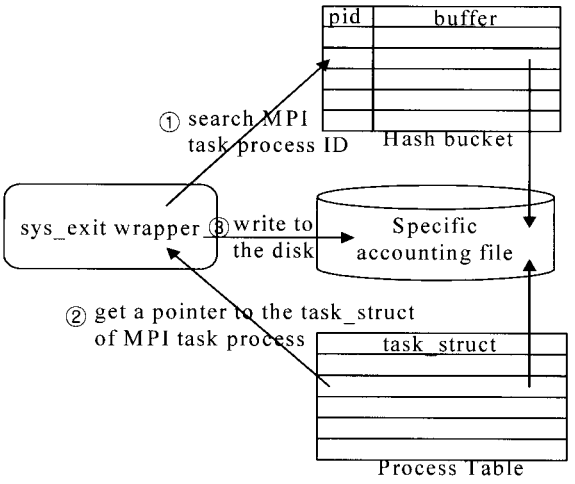
용하는 sys_accept 시스템 콜을 모니터링하여, 슬레이브 프로세스와 연결된 소켓 기술자의 아이노드 객체 번호로부터 /proc/net/tcp 파일을 검색하여 접속 요청을 허용하는 슬레이브 프로세스의 IP 주소와 포트 번호를 기록하였다. 이와 반대로, 슬레이브 프로세스에서는 마스터 프로세스에게 소켓 접속을 요청하는 sys_connect 시스템 콜을 모니터링하여 마스터 프로세스의 IP 주소와 포트 번호를 획득하였다.

4.3 로컬 어카운팅 데이터 수집

MPI 작업을 수행하는 프로세스들의 어카운팅 정보를 기록하기 위해 시스템 콜 랩퍼들을 커널 모듈에 추가시켜(그림 4), 각 시스템 콜 랩퍼에서 프로세스 어카운팅에 관련이 있는 데이터들을 버퍼에 일시적으로 보관하였다가 태스크 프로세스들이 종료될 때 버퍼에 저장된 데이터와 프로세스의 자원 사용량 데이터를 어카운팅 로그 파일에 저장하였다.

즉, sys_execve 시스템 콜 랩퍼에선 MPI 작업을 실행하기 위해 생성되는 프로세스들을 탐지하여, 해당 프로세스의 ID 및 작업 이름등을 버퍼에 저장하였고, 또한 sys_socketcall 시스템 콜 랩퍼에선 MPI Communicator에 참여하는 프로세스들간의 네트워크 접속 정보를 감지하여 버퍼에 저장하였다. 마지막으로 sys_exit 시스템 콜 랩퍼에서는, 감시 대상 프로세스들이 종료될 때마다 버퍼에 저장된 데이터와 프로세스 구조체로부터 추출된 자원 사용량 정보를 기록하였다.

각 노드의 로컬 어카운팅 시스템에서 병렬 작업을 수행하는 태스크 프로세스들의 자원 사용량을 측정하기 위한 데이터 구조체는 <표 2>와 같다. 먼저 CPU 사용량은 사용자 시간과 시스템 시간으로 나눠, 해당 프로세스의 tms 구조체를 참조하여 측정하였다. 그리고 프로세스의 가상 메모리를 관리하는 mm_struct 구조체를 참고하여 메모리 사용량을 측정하였다. 이 외의 다른 자원 사용량 데이터들도 해당 프로세서의 구조체를 참조하여 측정하였다(그림 6).



(그림 6) 프로세스 어카운팅 수집 과정

마스터 프로세스의 어카운팅 구조체 <표 3>은 해당 프로세스의 자원 사용량 데이터와 병렬 작업에 참여한 모든 슬레이브 프로세스의 자원 사용량을 취합하기 위한 부가적인 정보들로 구성하였다. 슬레이브 프로세스의 어카운팅 구조체 <표 4> 또한 공통적으로 측정한 자원 사용량 정보와 마스터 프로세스와의 네트워크 접속 정보로 구성하였다.

〈표 2〉 자원 사용량 데이터 구조체

```
#define   ACCT_COMM     16
#define   MAX_NP        32
typedef   __u16         comp_t;
struct acct {
    char      ac_flag;                  // accounting flags
    __u16     ac_uid;                   // user ID
    __u16     ac_gid;                   // group ID
    __u16     ac_tty;                   // controlling tty
    __u32     ac_btime;                 // beginning time
    comp_t    ac_utime;                 // user time
    comp_t    ac_stime;                 // system time
    comp_t    ac_etime                  // elapsed time
    comp_t    ac_mem;                   // average memory usage
    comp_t    ac_io;                    // characters transferred
    comp_t    ac_rw;                    // blocks read or written
    comp_t    ac_minflt;                // minor pagefaults
    comp_t    ac_majflt;                // major pagefaults
    comp_t    ac_swaps;                 // number of swaps
    __u32     ac_exitcode;              // process exit code
    char      ac_comm[ACCT_COMM+1];     // command name
};
```

〈표 3〉 마스터 프로세스의 어카운팅 데이터 구조체

```
struct master_acct {
  struct acct  ac_mac;                  // task accounting info.
  __u16        ac_pid;                  // process ID
  int          ac_np;                   // number of process
  char         ac_localaddr[14];        // local address and port
  char         ac_remoteaddr[MAX_NP][14];  // remote address and port
};
```

〈표 4〉 슬레이브 프로세스의 어카운팅 데이터 구조체

```
struct slave_acct {
    struct acct   ac_sac;               // task accounting info.
    __u16         ac_pid;               // process ID
    char          ac_localaddr[14];     // local address and port
    char          ac_remoteaddr[14];    // remote address and port
};
```

4.4 어큐뮬레이션 루틴

어큐뮬레이션 루틴은 각 클러스터 노드에 흩어져 있는 태스크 어카운팅 데이터들을 취합하여 병렬 작업 단위의 어카운팅 정보를 생산하여 DB에 저장하는 루틴이다. 어큐뮬레이션 루틴은 각 클러스터 노드에 있는 태스크 어카운팅 데이터를 수집하는 Robot, 각 노드에서 수집된 태스크 어카운팅 데이터를 합산하여 병렬 작업에 대한 전체 어카운팅 데이터를 만드는 Accumulator, 마지막으로 병렬 작업 단위로 어카운팅 데이터를 DB에 저장할 수 있도록 해주는 DB Connector로 구성되었다(그림 3).

Robot은 모든 클러스터 노드에 흩어져 있는 태스크 어카운팅 데이터를 수집하기 위해 MPI 프로그램으로 작성하였다. 따라서 각 로컬 어카운팅 시스템으로부터 측정된 어카운팅 데이터를 읽어 마스터 노드에 있는 프로세스에 전달한다. 그리고 마스터 노드에 있는 프로세스는 각 슬레이브 노드로부터 전달받은 어카운팅 데이터들을 노드별로 분리 저장하였다.

Robot은 모든 클러스터 노드에 흩어져 있는 각 로컬 어카운팅 데이터를 수집하기 위해 MPI 프로그램으로 작성하였다. 따라서 각 슬레이브 노드에서 처리되는 프로세스들은 로컬 어카운팅 데이터를 읽어 마스터 노드에 있는 프로세스에 전달한다. 그리고 마스터 노드에 있는 프로세스는 각 슬레이브 노드로부터 전달받은 어카운팅 데이터들을 노드별로 분리 저장하였다.

Accumulator는 마스터 어카운팅 데이터를 읽어, 마스터 프로세스와 MPI Communicator를 이루는 슬레이브 프로세스들의 어카운팅 데이터를 찾아 합산한 다음 DB Connector를 통해 DB에 저장한다. 이 때 마스터 어카운팅 데이터와 슬레이브 어카운팅 데이터들간의 네트워크 접속 정보(IP 주소와 포트번호)가 쌍방간에 일치하는지를 판단하여, MPI Communicator를 이루는 프로세스들의 집합들의 어카운팅 데이터를 합산하였다.

5. 실험 환경 및 결과

5.1 실험 환경

본 연구에서 구현한 Fork 로컬 스케줄러를 사용하는 리눅스 클러스터링 환경에서 MPI 작업의 자원 사용량 측정을 위한 어카운팅 시스템 개발 환경은 아래와 같다<표 5>. 그리고 MPI 작업의 태스크 할당 정책에 독립적인 자원 사용량 데이터를 측정하였다.

MPI 병렬 작업은 마스터 노드에서 mpirun 명령어를 통해 각 슬레이브 노드에서 처리될 태스크로 분할하여 전달한다. 이 때, MPI 병렬 작업을 구동시키는 mpirun 명령어는 기본적으로 관리자의 할당 정책 파일(machines.LINUX)을 참고하여, 각 슬레이브 노드로 태스크들을 분배한다. 또한 사용자가 정의한 할당 정책이 반영되도록 하기 위해서는 -machinefile이란 옵션을 이용한다.

〈표 5〉 실험 환경

항 목	내 용
환경	리눅스 클러스터 시스템 - 마스터 노드(1대) - 슬레이브 노드(3대) - NFS 서버(1대)
CPU, 메모리	866MHz, 256M / Node
운영체제 및 커널 버전	Red Hat Linux 7.1 Linux Kernel Version : 2.4.2-2
MPI 라이브러리 및 컴파일러	mpich 1.2.6 and gcc 2.96

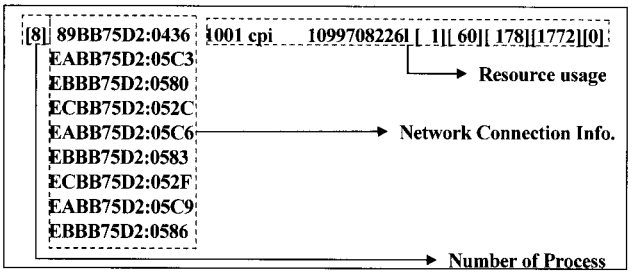
5.2 실험 결과 및 평가

MPI 라이브러리 기반 클러스터링 시스템 환경에서, MPI 병렬 작업 단위의 어카운팅 정보를 생산해 내는 로컬 어카운팅 시스템을 구현하였다. 모든 노드에 설치된 로컬 어카운팅 시스템은 태스크 프로세스들의 자원 사용량 정보와 이들과 협력하는 프로세스들간에 상호 네트워크 접속 정보를 기록한다. 마스터 노드의 로컬 어카운팅 시스템은 MPI 작업 전체의 총괄적인 제어를 담당하는 마스터 프로세스와 실질적인 작업 처리를 담당하는 슬레이브 프로세스들에 대한 정보를 수집한다. 슬레이브 노드의 로컬 어카운팅 시스템은 태스크 프로세스의 개별적인 자원 사용량 정보를 측정한다. 그리고 각 클러스터 노드에서 측정된 자원 사용량 정보가 태스크 할당 정책에 따라 어떤 영향을 미치는가에 대해 살펴본다.

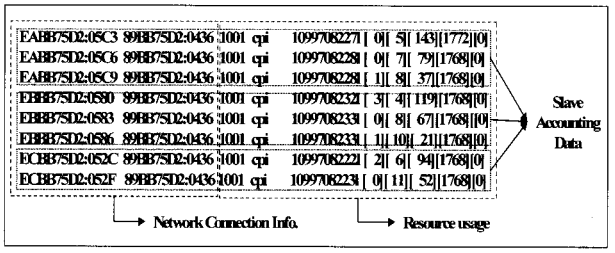
본 연구의 실험에서는 사용자가 작성한 MPI 병렬 작업에 대해 작업 어카운팅 데이터를 측정하였다. (그림 7)은 마스터 로컬 어카운팅 시스템에서 추출된 어카운팅 레코드의 한 예이다. 마스터 프로세스가 소비한 자원 사용량 데이터와 MPI Communicator내의 상호 협력하는 프로세스들의 IP 주소와 포트 번호를 보여주고 있다. 따라서 마스터 어카운팅 데이터에서 측정된 네트워크 접속 정보로부터, 식별 가능한 슬레이브 프로세스들을 찾아낼 수 있다.

(그림 8)은 각 슬레이브 로컬 어카운팅 시스템에서 기록된 어카운팅 파일로부터 (그림 7)의 마스터 프로세스와 동일 MPI Communicator내에 속한 슬레이브 프로세스들의 어카운팅 레코드이다.

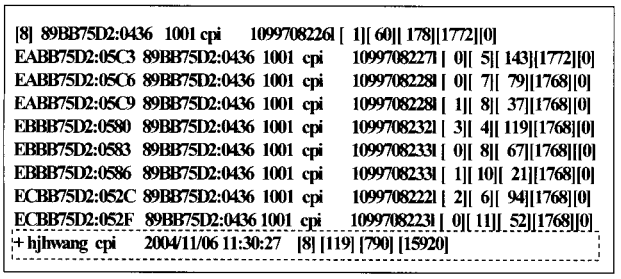
(그림 9)는 마스터 어카운팅 레코드와 슬레이브 어카운팅 레코드의 서로간의 네트워크 접속 정보가 일치하는 프로세스의 어카운팅 데이터를 추출하여, 병렬 작업을 처리하는데 모든 노드에서 소비한 자원 사용량 데이터를 합산한 결과이다.



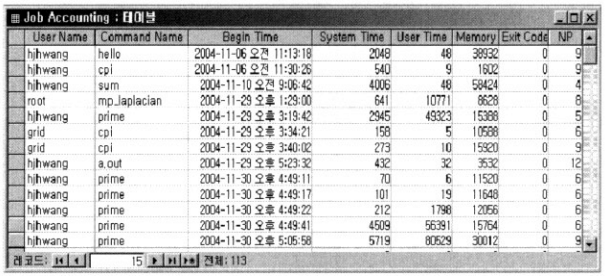
(그림 7) 마스터 노드의 어카운팅 레코드



(그림 8) 각 슬레이브 노드의 어카운팅 레코드



(그림 9) 어큘뮬레이션 루틴 수행 후, 어카운팅 데이터 병합


Job Accounting : 테이블

User Name	Command Name	Begin Time	System Time	User Time	Memory	Exit Code	NP
hjhwang	hello	2004-11-06 오전 11:13:18	2048	48	38932	0	9
hjhwang	cpi	2004-11-06 오전 11:30:26	540	9	1602	0	9
hjhwang	sum	2004-11-10 오전 9:06:42	4006	48	58424	0	4
root	mp_laplacian	2004-11-29 오후 1:29:00	641	10771	8628	0	8
hjhwang	prime	2004-11-29 오후 3:19:42	2945	49323	15388	0	5
grid	cpi	2004-11-29 오후 3:34:21	158	5	10568	0	6
grid	cpi	2004-11-29 오후 3:40:02	273	10	15920	0	9
hjhwang	a.out	2004-11-29 오후 5:23:32	432	32	3532	0	12
hjhwang	prime	2004-11-30 오후 4:49:11	70	6	11520	0	6
hjhwang	prime	2004-11-30 오후 4:49:17	101	19	11648	0	6
hjhwang	prime	2004-11-30 오후 4:49:22	212	1798	12056	0	6
hjhwang	prime	2004-11-30 오후 4:49:41	4509	56391	15764	0	6
hjhwang	prime	2004-11-30 오후 5:05:58	5719	80529	30012	0	9

레코드: 15 전체: 113

(그림 10) MPI 작업 자원 사용량 정보

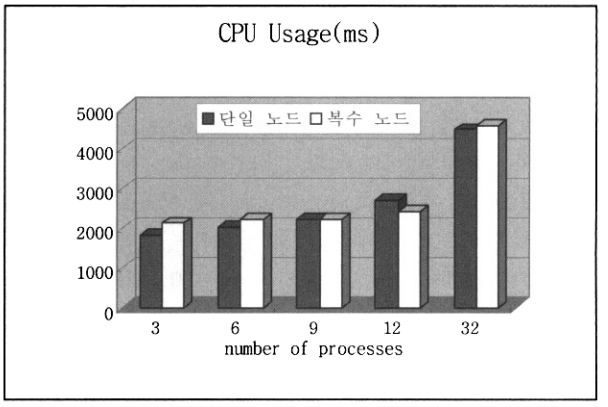
(그림 10)은 (그림 9)에서 취합한 작업 어카운팅 데이터들을 관리하기 위해 DB에 저장된 결과를 보여주고 있다. (그림 9)와 비교해서 특이한 점은 동일 병렬 작업에 참여한 프로세스들의 자원 사용량 데이터들만 DB에 저장되어 있고, 네트워크 접속 정보는 빠져 있다는 것을 알 수 있다. 왜냐하면 사용자에게는 불필요한 정보일 뿐만 아니라, 시스템 내부의 보안적인 문제가 야기될 수 있기 때문이다. 따라서 어카운팅 서비스가 이루어질 때는 사용자에게 필요한 최소한의 정보를 제공하는 것이 바람직하다.

이번에는 균등 분배 MPI 작업을 태스크로 분할한 후, 이들 태스크들을 한 노드에 할당한 경우와 모든 슬레이브 노드에 균등하게 분배함에 따라 동일 MPI 병렬 작업의 자원 사용량이 태스크 할당 정책에 따라 어떠한 영향을 미치는지 측정하였다.

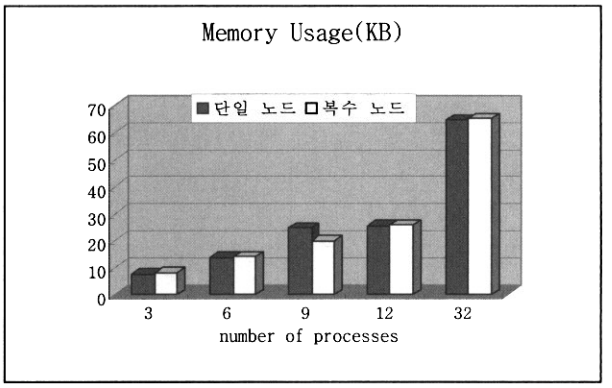
(그림 11)은 모든 태스크들을 한 노드에 할당한 경우와 복수개의 노드에 균등하게 할당한 경우, 동일 MPI 작업을 처리하는데 소비한 전체 CPU 사용량은 비슷하게 측정되었다. 그리고 (그림 12)에서도 각 태스크 프로세스들이 사용한 평균 메모리 사용량도 비슷하게 측정되었다.

따라서 이 실험을 통해 알 수 있듯이 태스크 할당 정책에 큰 영향을 받지 않고, 동일 MPI 작업에 대해 단일 노드에서 처리하거나 여러개의 복수 노드에서 균등 처리하든지간에 동일 MPI 작업을 수행하는데 소비한 전체 자원 사용량은 비슷하게 측정되었다. 그리고 동일 작업에 대해, 하나의 프로세스가 처리하는데 소비한 자원 사용량은 동일 시스템 환경에서 대단위의 프로세스들로 나눠서 처리하는데 드는 자원 사용량의 합과 비슷함을 알수 있었다. 그러나 실행 시간 측면에서는 단일 노드에서 보다는 여러 개의 복수 노드에서 동시에 처리하는 것이 매우 효과적인 것을 알 수 있었다.

본 논문에서 개발한 MPI 환경에서 자원 사용량 측정을 위한 어카운팅 시스템의 실험 결과에서 알 수 있듯이 사용



(그림 11) 할당 정책에 따른 CPU 사용량



(그림 12) 할당 정책에 따른 Memory 사용량

자 병렬 작업을 처리한 모든 프로세스의 자원 사용량 정보를 측정할 수 있었다. 따라서 Fork 로컬 스케줄러를 사용해서 MPI 병렬 작업을 처리하는 클러스터 환경에서 전통적인 로컬 어카운팅 시스템으로는 병렬 작업의 어카운팅 데이터를 측정할 수 없지만, 본 논문에서 제안한 어카운팅 시스템은 병렬 작업에 대해서도 어카운팅 정보를 제공할 수 있게 된다. 따라서 본 논문에서 제안한 어카운팅 시스템도 MPI 병렬 작업에 대해 LoadLeveler, LSF, PBS와 같이 어카운팅 서비스가 가능하며, 그리드 어카운팅 시스템에도 활용 가능할 것으로 평가된다.

6. 결 론

현재 유닉스 계열의 전통적인 로컬 어카운팅 시스템은 단일 노드상의 프로세스 자원 사용량 정보만을 제공하기 때문에, 클러스터링 환경에서 MPI 병렬 작업에 대한 어카운팅 정보 서비스를 제공할 수 없다. 각 노드에서 동일 병렬 작업을 처리하기 위해 상호 협력하는 프로세스들간의 어카운팅 정보를 취합할 수 있는 정보가 부족하기 때문이다.

본 논문에서는 클러스터링 환경에서 MPI 병렬 작업 단위의 어카운팅 정보 서비스가 가능한 로컬 어카운팅 시스템을 구현하였다. 각 클러스터 노드에 설치된 로컬 어카운팅 시스템은 시스템상에 실행되는 모든 프로세스들의 활동을 추적하여, MPI 병렬 작업을 수행하는 프로세스가 종료할 때 프로세스 자원 사용량을 기록하였다. 또한 로컬 어카운팅 시스템은 병렬 작업 어카운팅 정보를 취합할 수 있도록 하기 위해 MPI 작업을 수행하는 프로세스들간의 상호 네트워크 접속 정보를 기록하였다. 각 클러스터 노드의 어카운팅 파일에서 수집된 네트워크 접속 정보를 토대로 프로세스 자원 사용량을 취합하여 병렬 작업 단위의 어카운팅 정보를 생산하였다.

따라서 본 논문에서 제안한 어카운팅 시스템을 통해서 측정된 자원 사용량 정보가 LoadLeveler, PBS등과 같은 로컬 스케줄러에서 제공된 프로세스 자원 사용량 정보와 동일함을 알 수 있다. Fork 로컬 스케줄러를 사용하는 MPI 라이브러리 기반 클러스터링 환경 하에서 병렬 작업에 대한 자원 사용량을 측정할 수 있어 어카운팅 정보 서비스가 가능하다. 그리고 어카운팅 시스템으로부터 측정된 자원 사용량 정보는 요금 부과 알고리즘을 거쳐 사용자에게 과금 서비스가 가능하다.

또한 개발된 시스템의 간단한 조작만으로 UR-WG에서 제안한 자원 사용량 데이터 항목들을 측정할 수 있다. 그러므로 지리적으로 분산된 이질적인 자원을 통합한 그리드 컴퓨팅 환경의 그리드 어카운팅 시스템에서 활용할 수 있다.

그리드에 참여하는 각 사이트에서는 로컬 스케줄러나 로컬 어카운팅 시스템으로부터 측정된 어카운팅 데이터를 기초로 다양한 어카운팅 정책을 펼 수 있다. 그리고 그리드가 상용화되면 여러 어카운팅 정책에 따라 자원 제공자와 자원 소비자간의 다양한 형태의 경제 모델도 등장할 것이다. 그러므로 다양한 어카운팅 정책에 따라 그리드 비즈니스를 활성화하기 위해선 다양한 종류의 자원 소비 항목들을 찾아 제공되어져야 할 것이다.

참 고 문 헌

[1] M. Baker and R. Buyya, "Cluster Computing at a Glance", High Performance Clustering Computing Vol 1, Pretice Hall, 1999.

[2] Anthony Beardsmore, Keith Hartley, et al, "GSAX(Grid Service Accounting Extensions)," GGF OGSA Resource Usage Service Working Group, 2002.

[3] Sebastian Ho, "GridX System Design Documentation," Bioinformatics Institute, Singapore, 2002.

[4] Rodney Mach, "Accounting Interchange Natural Language Description(Requirements)," GGF Usage Record Working Group, 2004.

[5] IBM LoadLeveler for AIX 5L, Using and Administering, version 3, release 1, 2001.

[6] PBS(Portable Batch System), http://www.openpbs.org

[7] Ian Foster, Carl Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Mogan Kaufmann Publishers, 1998.

[8] The DataGrid Project, http://eu-datagrid.web.cern.ch/eu-datagrid

[9] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International J. Supercomputer Applications, 15(3), 2001.

[10] I. Foster, C. Kesselman, "Globus : A Metacomputing Infrastructure Toolkit," International J. Supercomputer Applications, 11(2), pp.115-128, 1997.

[11] Karl Czajkowski, Ian Foster, et al, "A Resource Management Architecture for Metacomputing Systems," Procs. of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[12] The Message Passing Interface(MPI) standard, http://www-unix.mcs.anl.gov/mpi

[13] W. Richard Stevens, "Unix Network Programming", Prentice Hall, 1990.

[14] Linux Documentation Project, http://www.tldp.org

[15] Linux BSM, http://sourceforge.net/projects/linuxbsm

[16] SNARE, http://intersectalliance.com

황 호 전

e-mail : hjhwang@mail.chonbuk.ac.kr
1997년 전북대학교 컴퓨터공학과(공학사)
1999년 전북대학교 컴퓨터공학과(공학석사)
1999년~현재 전북대학교 컴퓨터공학과 박사과정
관심분야 : 분산 및 병렬처리, 그리드

안 동 언

e-mail : duan@moak.chonbuk.ac.kr
1981년 한양대학교 전자공학과(공학사)
1987년 KAIST 전산학과(공학석사)
1995년 KAIST 전산학과(공학박사)
1995년~현재 전북대학교 전자정보공학부 부교수
2001년~2002년 전북대학교 정보검색시스템연구센터 센터장
관심분야 : 정보검색, 한국어정보처리, 문서분류, 문서요약

정 성 종

e-mail : sjchung@moak.chonbuk.ac.kr
1975년 한양대학교 전기공학과(공학사)
1981년 Houston대학교 전자공학과(공학석사)
1988년 충남대학교 전산공학과(공학박사)
1985년~현재 전북대학교 전자정보공학부 교수
1996년~1998년 전북대학교 전자계산소 소장
2001년~현재 전북대학교 BK21 전자정보사업단 단장
관심분야 : 정보검색, Grid