韓國 컴퓨터情報學會 論文誌
第11卷 第1號, 2006. 3.

2006-11-1-1-9

계산 그리드 상에서 프로그램의 특성을 반영한 작업 프로세스 수의 결정에 관한 연구

조 수 현*, 김 영 학**

A Study on Determination of the Number of Work Processes Reflecting Characteristics of Program on Computational Grid

Soo-Hyun Cho*, Young-Hak Kim**

요 약

계산 그리드 환경은 서로 다른 성능과 이질적인 네트워크 상태들을 갖는 LAN/WAN으로 구성되고 다양한 형태의 프로그램이 수행된다. 이러한 환경에서 각 노드의 작업은 프로그램의 특성에 따라 이질적인 네트워크 환경과 각 노드의 컴퓨팅 파워를 고려하여 수행되기 때문에 자원 선택 브로커의 역할은 매우 중요하다. 본 논문은 계산 그리드 환경에서 프로그램 특성에 따라 네트워크 상태 정보와 각 노드의 성능을 고려하여 각 노드에 할당될 작업 프로세스의 수를 결정하는 새로운 자원 선택 브로커를 제안한다. 제안된 자원 선택 브로커는 다음과 같이 3단계로 구성된다. 첫째, 프로그램의 특성을 반영하여 지연시간, 대역폭 정보와 cpu 혼합정보를 이용하여 각 노드의 성능비율을 계산하고, 이러한 비율에 의해 각 노드에서 수행될 작업 프로세스의 수를 결정한다. 둘째, 이전 단계에서 결정된 작업 프로세스의 수를 기반으로 RSL 파일을 자동으로 생성한다. 마지막으로, 각 노드는 RSL 파일을 이용하여 작업 프로세스들을 생성하고 자신에 할당된 작업을 수행한다. 실험 결과에 의하면 작업량, 프로세스 수, 노드 수 관점에서 프로그램의 특성을 반영한 제안된 방법이 기존 방법(균등)과 지연시간-대역폭을 고려한 것에 비해 278%~316%, 524%~595%, 924%~954% 향상되었다.

Abstract

The environment of computational grid is composed of the LAN/WAN each of which has different efficiency and heterogeneous network conditions, and where various programs are running. In this environment, the role of the resource selection broker is very important because the work of each node is performed by considering heterogeneous network environment and the computing power of each node according to the characteristics of a program. In this paper, a new resource selection broker is presented that decides the number of work processes to be allocated at each node by considering network state information and the performance of each node according to the characteristics of a program in the environment of computational grid. The proposed resource selection broker has three steps as follows. First, the performance ratio of each node is computed using latency-bandwidth-cpu mixture information reflecting the characteristics of a program, and the number of work processes that will be performed at each node are decided by this ratio. Second, RSL file is automatically made based on the number of work processes decided at the previous step. Finally, each node creates work processes by using that RSL file and performs the work which has been allocated to itself. As experimental results, the proposed method reflecting characteristics of a program, compared with the existing (uniformity) and latency-bandwidth method is improved 278%~316%, 524%~595%, 924%~954% in the point of work amount, work process number, and node number respectively.

▸ Keyword : Computational Grid, Number of Work Processes, Resource Selection Broker, Characteristics of a Program, RSL

• 제1저자 : 조수현
• 접수일 : 2006.01.03, 심사완료일 : 2006.02.05
* 금오공과대학교 컴퓨터공학부 계약교수, ** 금오공과대학교 컴퓨터공학부 부교수
※ 본 연구는 2005년도 금오공과대학교 학술연구비 지원에 의해 수행되었습니다.

Ⅰ. 서 론

계산 그리드(computational grid)는 지리적으로 분산되어 있는 고성능 컴퓨팅 자원을 네트워크로 상호 연동하여 조직과 지역에 관계없이 가상 집합체들의 자원을 공유하여 방대한 계산 작업을 수행할 수 있는 기술이다. 최근에 나노기술과 생명과학으로 대변되는 21세기 첨단 과학기술 시대에는 천문학적인 정보 트래픽, 방대한 저장 공간, 강력한 계산능력 등이 네트워크를 통하여 공유될 것이고, 이를 실현하는 유력한 수단으로 그리드 컴퓨팅이 주목받고 있다.

LAN/WAN(local area network/wide area network)으로 구성된 계산 그리드는 각 자원들의 성능과 네트워크 상태가 이질적인 환경에 놓여 있다. 그래서 연결된 노드들의 컴퓨팅 파워뿐만 아니라, 네트워크를 통한 작업의 송수신이 이루어지기 때문에 무엇보다 네트워크 상태정보들을 고려하여 계산 작업에 반영시키는 것이 성능향상에 중요하며 이를 위해서는 일반 응용 프로그램 사용자들에게는 많은 노력들이 요구된다. 그리고 계산 그리드 환경에서 수행되는 프로그램의 특성에 따라 네트워크 성능정보와 노드별 컴퓨팅 파워 정보들의 적용 유무에 따라 성능이 좌우되기에 반드시 고려되어야 한다.

그리드 컴퓨팅을 지원하는 대표적인 미들웨어인 글로버스(globus)[1] 툴 킷은 검색되어진 자원에 대한 선택과 할당에 관한 기능을 제공하고 있다. 하지만 사용자는 MDS(metacomputing directory service)를 이용하여 자원 및 정보이용에 있어 수동적으로 이용하는 단계에 불가하다. 즉 사용자의 노력으로 노드에 대한 위치 및 일반적인 정보(cpu, memory)만을 이용할 뿐 정작 계산 그리드 환경에서 성능향상에 중요한 영향을 미치는 네트워크 상태정보와 그에 따른 노드 분석 및 분류 기능 등의 자동화 도구[2,3]가 제공되지 않고 있다. 또한 계산 그리드 상에서 응용 프로그램의 특성을 고려한 작업 프로세스 수를 결정하는 방법에 대해서도 고려되지 않고 있다.

본 논문에서는 계산 그리드 상에서 응용 프로그램의 특성에 따라 사용자가 요구하는 작업 프로세스 수가 전체 수행시간에 어떤 영향을 미치는지 살펴본다. 본 논문에서는 응용 프로그램의 특성을 네트워크 의존적인 프로그램과 계산 의존적인 프로그램으로 나누어 평가한다. 네트워크 의존적인 프로그램은 네트워크를 통해 작업을 송수신하기에 오히려 작업 프로세스 수가 많아지면 각 프로세스 간의 통신시간이 증가되어 결국 전체 성능이 저하된다. 그래서 사용자가 요구한 전체 프로세스 수를 갖고 계산 작업에 참여하는 것보다 최적의 작업 프로세스 수를 결정하여 적은 수의 프로세스를 갖고 계산 작업에 참여하여 프로세스 간 통신 시간을 절약함으로써 성능향상을 얻을 수 있다. 또한 계산 의존적인 프로그램인 경우 상대적으로 네트워크 성능정보보다 실제 계산 작업에 영향을 끼치는 노드별 cpu 자원을 계산 작업에 반영 시킴으로써 전체 수행시간을 단축시킬 수 있다.

따라서 본 논문에서는 이더넷 환경의 저속, 고속 네트워크 망에 연결된 PC 자원을 NWS(network weather service)[4]를 이용하여 네트워크 성능정보(latency, bandwidth)와 노드별 사용가능한 cpu 정보를 기반으로 프로그램 특성을 고려하여 노드별 작업 프로세스 수를 결정하는 방법을 제안한다. NWS로부터 수집된 정보를 기반으로 노드별 분석 및 분류작업을 통해 프로그램 특성을 고려한 노드별 성능비율을 계산한 후 생성될 작업 프로세스 수를 결정한다. 최종적으로, 결정된 작업 프로세스 수를 통해 각 노드에 수행할 정보를 갖고 있는 RSL(resource specification language)[5] 파일을 자동으로 생성시켜 수행하는 자원 선택 브로커(resource selection broker)를 구현하였다. 실험결과, 응용 프로그램의 특성에 따라 네트워크 정보와 노드별 cpu 정보를 적절히 적용하여 각 노드에 생성될 작업 프로세스 수를 결정했을 경우 그렇지 않은 기존방법보다 큰 폭으로 성능이 향상됨을 알 수 있었다.

본 논문의 구성은 다음과 같다. II장에서는 관련연구에 대해 개괄적으로 설명하고, III장은 제안된 프로그램 특성을 고려한 노드별 작업 프로세스 수 결정방법과, IV장은 시스템 구조에 대해 언급한다. V장에서는 실험결과 및 분석에 대해서, 끝으로 VI장에서 결론 및 향후 연구를 기술한다.

Ⅱ. 관련연구

사용자의 노력으로 글로버스 내 MDS 서버를 이용하여 노드들의 성능정보(cpu, memory)들을 계산 작업에 반영시킬 수 있다[6]. MDS는 글로버스를 이용한 그리드 컴퓨팅

에 참여한 노드들의 성능정보를 관리하는 서버이다. 즉 사용자는 참여한 노드들의 성능정보를 요청하여 해당 정보를 받아 사용할 수 있다. 제공받은 성능정보를 갖고 노드별 계산 작업량을 차등화하여 전체 수행시간을 줄이는 방법이다[7]. 하지만, 제공받은 성능정보가 노드들의 컴퓨팅 파워에 국한되기 때문에 실제 계산 작업에서의 성능 향상에 많은 한계점이 있다. 또한 계산 그리드 환경에서의 계산 작업에 있어 중요한 요소인 네트워크에 대한 성능정보가 제공되지 않으며 사용자가 요구하는 다양한 정보에 대한 수집 및 분류 기능 등도 포함하고 있지 않다. 프로그램의 특성에 따라 네트워크 성능정보와 노드별 컴퓨팅 파워를 고려하여 효율적인 노드 선택과 작업 프로세스 수를 결정하는 브로커 역할에 대해서도 취약하다[8],[9]. 그래서 글로버스를 통해 작업을 수행하려면 현재 자원의 위치를 직접 알아서 수동으로 RSL 파일을 작성해야 하는 불편함이 있다.

Condor-G[10], Nimrod-G[11]에서는 리소스 브로커를 구현하고 있다. 하지만 응용 프로그램에서 원하는 메모리, cpu(cpu수와 load)에 대한 리소스를 찾아 자원을 할당하여 작업을 처리하는 방법에 국한되어 있을 뿐, 본 논문에서 제안하는 전체 수행시간 단축을 위해 작업 프로세스 수를 결정하는 방법과 네트워크 성능정보에 대해서는 고려하지 않고 있다. 또한 기존의 Condor와 연동해서 사용되어야 하는 불편함을 갖고 있다.

네트워크 성능정보를 고려한 토폴로지 구성 방법은 계산 작업에 참여한 노드별 성능정보를 NWS 등을 이용하여 정보를 수집한 후, 성능정보에 따른 가상 토폴로지(topology)를 구성하여 전체 계산 시간을 향상시키는 방법이다[12],[13]. 하지만 네트워크 성능정보 수집과 가상 토폴로지 구성작업이 응용 프로그램이 시작되는 시점에서 이뤄진다. 즉 전체 수행시간에서 성능정보 측정 및 수집시간과 토폴로지 구성 시간 등이 불필요하게 반영됨으로 성능향상에 많은 한계점이 있다.

NWS를 이용한 그리드 계산 방법은 글로버스 내 MDS 서버보다 많은 성능정보를 제공한다. 특히 네트워크 상태정보를 이용할 수 있다[14]. NWS에서 제공되는 네트워크 성능정보와 실제 그리드 환경에서의 계산 작업간의 아무런 관련성이 없으므로 사용자에게는 이 같은 연동 작업에 대한 많은 노력들이 요구된다. 즉 각 성능정보에 따른 분류작업과 노드별 작업 프로세스 수를 결정하여 노드별로 수행할 정보를 갖고 있는 RSL 파일을 자동으로 만드는 기능 등이 포함되지 않고 있다. 그리고 기존 대부분의 연구에 대한 성능평가는 WAN 기반의 전용 고속 네트워크 망으로 구성된 그리드 환경이며 저속의 네트워크 망과 PC 자원에 대한 실험평가도 미흡하다.

따라서 본 논문에서는 저속의 이질적인 네트워크 상태와 PC 자원의 환경에서 프로그램 특성을 고려하여 네트워크 성능정보와 노드별 cpu 정보를 계산 작업에 반영하여 각 노드의 작업 프로세스 수를 결정하는 방법을 제안한다. 또한 최종적으로 결정된 작업 프로세스 수를 통해 각 노드에 수행할 정보를 갖고 있는 RSL 파일을 자동으로 생성하여 수행하는 자원 선택 브로커를 구현하였다.

III. 프로그램 특성을 반영한 작업 프로세스 수 결정 방법

본 절에서는 계산 그리드 환경에서 프로그램 특성을 고려하여 사용자가 요청한 작업 프로세스 수를 결정하는 방법을 살펴본다. 프로그램 특성으로는 네트워크 의존적인 경우와 계산 의존적인 프로그램으로 나누어 평가한다. 네트워크 의존적인 프로그램인 경우에는 네트워크를 통한 계산 작업 시 작업을 수행하는 프로세스들의 수가 많아지면 오히려 그들 간의 데이터 송수신을 위한 통신시간이 증가하여 전체 성능이 저하된다. 또한 계산 의존적인 프로그램은 네트워크의 소요시간보다 각 노드에 의한 계산시간이 상대적으로 큰 비중을 차지하기 때문에 cpu 자원을 고려하여 작업 프로세스 수를 결정해야한다. 따라서 프로그램의 특성을 고려하여 네트워크 성능지표(latency+bandwidth)와 사용가능한 cpu 정보를 기반으로 사용자가 요청한 작업 프로세스 수를 현 네트워크와 참여한 노드들의 cpu 성능에 적합한 프로세스 수를 결정하는 방법을 살펴본다.

3.1 전체 수행시간 분석

본 논문에서 수행하는 계산 모델은 마스터/슬레이브 모델이다. 계산 그리드 환경에서의 마스터 노드에 생성된 마스터 프로세스(이하 마스터)는 서브 노드에 생성된 작업 프로세스(이하 슬레이브)들로의 작업 배분과 결과 값을 취합하고 슬레이브들은 마스터로부터 전송받은 작업을 수신하여 처리한 후 마스터로 결과 값을 전송한다. 전체 수행시간 계산 방법은 마스터에서 슬레이브들로 작업들을 전송한다. 마스터는 슬레이브가 처리한 결과 값을 수신하여 취합 과정을

최종 완료하는 시점을 전체 수행시간으로 간주한다. 다음은 전체 수행시간 분석에 사용된 용어를 정의한다.

- P : 사용자가 요청한 작업 프로세스 수를 의미한다.
- α : 노드별로 생성된 프로세스들로 작업 전송 시 소요되는 네트워크 시간을 의미한다.
- β : 노드별로 생성된 프로세스들이 수신된 작업을 처리하는 시간을 의미한다.
- t_{send} : 작업을 보내는 컴퓨터에서 작업 전송을 위해 준비되는 소프트웨어적인 지연시간을 의미한다.
- t_{recv} : 수신 컴퓨터에서 작업을 수신 받기 위한 소프트웨어적인 오버헤드를 의미한다.
- t_{net} : 네트워크를 통해 작업이 전송되기 위해 지연되는 시간을 의미한다. 일반적으로 네트워크의 대역폭(bandwidth), 스위칭 기법(switching mechanism), 블록킹 시간(blocking time) 등의 요소들이 작업 전송시간에 중요한 영향을 주게 된다.
- t_{hold} : 연속적인 작업 전송에 있어 다음 작업 전송을 위해 소요되는 지연시간을 의미한다.
- $t_{net\_send\_to\_slave}$: 슬레이브로의 작업 전송 시 소요되는 네트워크 시간을 의미한다.
- $t_{net\_recv\_from\_slave}$: 슬레이브로부터 계산된 결과 값을 수신할 때 소요되는 네트워크 시간을 의미한다.
- t_{master_cpu} : 슬레이브로부터 계산된 결과 값을 취합하는 시간을 의미한다.
- t_{slave_cpu} : 슬레이브에서 수신된 작업을 처리하는 시간을 의미한다.

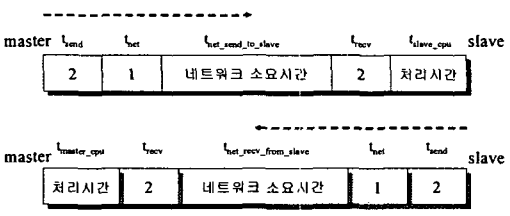


그림 1. 전체 수행시간 분석
Fig 1. The total execution time analysis

$$t_{total} = (t_{send} \times (P-1) \times 2) + (t_{recv} \times (P-1) \times 2) + (t_{net} \times (P-1) \times 2) + (t_{net\_send\_to\_slave}) + (t_{net\_recv\_from\_slave} \times (P-1)) + (t_{slave\_cpu}) + (t_{master\_cpu} \times (P-1)) + (t_{hold} \times (P-2)) \cdots\cdots (식\ 1)$$

$$t_{net\_send\_to\_slave} = (P-1) * \alpha$$
$$t_{slave\_cpu} = (P-1) * \beta$$

계산 작업에 참여한 작업 프로세스 수를 P 라고 할 때 전체 수행시간은 (그림 1)과 같이 표현할 수 있다. 작업 전송 지연시간(2ms), 결과 값 수신 지연시간(2ms), 네트워크 지연시간(1ms), 연속적인 전송 간격시간(2ms)이 일정하다고 가정한다. 따라서 (그림 1)과 같이 전체 수행시간은 마스터에서 작업을 슬레이브로 전달되는 네트워크 소요시간과 슬레이브가 수신된 작업을 처리하는 시간, 슬레이브에서 계산된 결과 값이 네트워크를 통해 마스터로 전송되는 시간, 마스터에서 슬레이브로부터 계산된 결과 값을 취합하는 시간에 따라 전체 성능이 좌우된다는 것을 알 수 있다.

식(1)은 전체 수행시간을 수식으로 표현한 것이다. 한 노드에 생성된 하나의 프로세스는 작업에 참여하지 않고 요청 및 결과 값만을 확인하는 순수 클라이언트이므로 작업에 참여하는 프로세스 수는 P-1이 된다. 슬레이브로의 작업 전송 시 네트워크 소요시간은 작업에 참여하는 프로세스 수에 프로세스별로의 네트워크 소요시간(α)의 곱으로 표현할 수 있다. 물론 프로세스별로의 네트워크 소요시간은 네트워크 상황에 따라 가변적으로 변화한다. 노드별 프로세스들의 처리시간은 참여 프로세스 수에 프로세스별 처리시간의 곱으로 나타낸다. 프로세스별 처리시간(β)은 노드별 성능에 따라 가변적으로 변화한다. 따라서 전체 수행시간 중 프로그램의 특성에 따라 네트워크로 전송되는 시간과 노드별로 처리하는 시간의 비중이 다르기 때문에 자원 선택 시 반드시 고려해야 할 사항이다.

네트워크 의존적인 프로그램인 경우 작은 데이터들이 프로세스 간의 통신에 의해 수행되기 때문에 노드들의 성능과 별개로 노드별로 처리하는 시간, 계산된 결과 값이 마스터로 전송되는 네트워크 시간, 마스터에서 계산된 결과 값을 취합하는 시간은 일정하여 전체 성능에 영향을 미치지 않는다. 그래서 마스터에서 슬레이브로 전달되는 네트워크 소요시간이 가변적으로 변화하기 때문에 이를 계산 작업에 반영시켜야 한다. 즉 네트워크 소요시간에 영향을 미치는 지연시간(latency), 대역폭(bandwidth)이 전체 수행시간에 많은 부분 반영이 되기 때문에 반드시 계산 작업에 고려되어야 함을 알 수 있다.

계산 의존적인 프로그램은 네트워크로 각 노드의 계산된 결과 값을 수신하는 시간과 마스터에서 결과 값을 취합하는 시간은 일정하지만 작업 전송과 노드별 처리하는 작업량이 상대적으로 크기 때문에 네트워크 상태정보와 노드별 cpu 정보가 중요한 영향을 미친다. 따라서 자원 선택 브로커는 전체 수행시간을 단축시키기 위해서 슬레이브로 전달되는 네트워크 소요시간과 노드별 처리시간을 고려하여 계산 작업에 반영시키는 것이 무엇보다 중요하다. 즉 프로세스 간 네트워크 소요시간을 단축시키기 위해서는, 네트워크 상태정보를 고려하고 노드별 처리시간을 향상시키기 위해서는 노드별 cpu 성능정보를 함께 고려해야 함을 알 수 있다.

3.2 네트워크 의존적인 프로그램

계산 그리드 환경에서의 네트워크 의존적인 프로그램은 계산 작업 시 슬레이브들이 수신된 작업을 처리하는 시간보다 마스터에서 슬레이브들로 작업을 전송 및 결과 값을 수신할 때 네트워크 상에서 소요되는 시간이 보다 많은 비중을 차지하는 프로그램을 말한다. 즉 노드별 cpu 자원에 의존적이지 않고 프로세스 간 통신시간에 따라 전체 성능이 좌우된다. 그래서 전체 수행시간 중 네트워크 상태에 중요한 영향을 끼치는 통신 지연시간과 대역폭을 고려하여 노드별 작업 프로세스 수를 결정해야 한다. 다음은 네트워크 의존적인 프로그램 상황에서 노드별 작업 프로세스 수를 결정할 때 사용되는 용어 및 수식을 나타낸다.

- total_latency_normal_sum : 각 노드들의 지연시간 값에 대한 성능 값의 총합을 의미한다.
- total_bandwidth_sum : 전체 노드들의 대역폭 값의 총합을 의미한다.
- Ni_normal_value : 노드 i의 지연시간 역수에 대한 정규화 값을 의미한다.
- Ni_bandwidth : 노드 i의 측정된 대역폭 값을 의미한다.
- Ni_latency_bandwidth_sum : 노드 i의 지연시간, 대역폭을 혼합한 성능비율의 합을 의미한다.
- Ni_bandwidth_percentage : 노드 i의 대역폭에 대한 성능비율 값을 의미한다.
- Ni_latency_percentage : 노드 i의 지연시간에 대한 성능비율 값을 의미한다.
- Ni_latency_bandwidth_percentage : 노드 i의 지연시간, 대역폭을 혼합한 성능비율 값을 의미한다.
- Ni_latency_bandwidth_process : 지연시간-대역폭을 고려하여 노드 i에 생성될 작업 프로세스 수를 의미한다.
- Ni_network_process : 네트워크 의존적인 프로그램 수행 시 노드 i에 생성될 작업 프로세스 수를 의미한다.

- Ni_latency_percentage = Ni_normal_value / total_latency_normal_sum (식 2)

- Ni_bandwidth_percentage = Ni_bandwidth / total_bandwidth_sum ····· (식 3)
- Ni_latency_bandwidth_sum = Ni_latency_percentage + Ni_bandwidth_percentage

- Ni_latency_bandwidth_percentage = Ni_latency_bandwidth_sum / C
  Ni_latency_bandwidth_process = ROUND (P * Ni_latency_bandwidth_percentage) ····· (식 4)
  Ni_network_process = ROUND (P * Ni_latency_bandwidth_percentage * weight) (식 5)

  weight >= (N-1) / P, if not : 각 노드당 하나의 프로세스를 생성

사용자가 요청한 프로세스 수를 P라고 할 때, 먼저 측정된 노드별 지연시간(식(2))에 대한 성능비율을 구하고, 대역폭(식(3)) 또한 성능비율을 계산하여 식 (4)와 같이 지연시간과 대역폭에 대한 성능비율을 혼합하여 노드별 작업 프로세스 수를 결정한다. 혼합한 성능요소의 수를 C라고 하면 본 논문에서는 지연시간, 대역폭을 혼합하였기에 2가 된다. 지연시간과 대역폭의 성능비율 합을 C로 나누면 해당 노드의 지연시간, 대역폭을 혼합한 성능비율이 계산된다. 따라서 요청한 프로세스 수에 혼합한 성능비율을 적용하면 해당 노드의 작업 프로세스 수가 결정된다[15].

하지만 식 (4)의 프로세스 수는 사용자 요청에 의한 전체 작업 프로세스 수를 지연시간, 대역폭 혼합 정보를 기반으로 노드별 작업 프로세스 수를 결정한 것이다. 예를 들면 참여한 노드 수가 3개이고 전체 작업 프로세스 수를 12개 요청했을 경우 계산된 노드별 성능비율이 20%, 30%, 50% 일 때 2, 4, 6개 씩 작업 프로세스들이 노드별로 생성된다. 지연시간, 대역폭을 고려하여 식 (4)에 의해 노드별 작업 프로세스 수를 결정하면 균등하게 작업 프로세스를 생성하는 기존방식보다 어느 정도 성능이 향상된다. 하지만 사용자가 요청한 12개의 프로세스 모두를 성능요소에 따라 노드별로 생성시켜 작업을 수행하기 때문에 전체 수행시간

중 여전히 네트워크에서 소요되는 시간이 많아 성능 향상에 한계가 있다.

따라서 본 논문에서는 계산 그리드 환경에서 네트워크 의존적인 프로그램을 수행할 때 작업 프로세스 수를 감소시켜 그로 인한 네트워크 소요시간을 줄임으로써 전체 수행을 단축시킬 수 있는 방법을 제안한다. 프로세스 수를 줄이기 위해 본 논문에서는 노드별로 가중치(weight) 값을 적용하였다. 가중치의 의미는 네트워크 상태정보(지연시간+대역폭)를 고려하여 결정된 노드별 프로세스 수에 가중치 값을 적용하여 각 노드의 프로세스 수를 감소시키는 것을 뜻한다. 예를 들면 가중치 값이 20%이고 네트워크 상태정보를 고려하여 노드 A에 결정된 프로세스 수가 4개인 경우 가중치 20% 값을 곱하면 해당 노드에 생성되는 프로세스 수가 계산된다(4*0.2=0.8≒1개). 즉, 노드 A는 기존 4개에서 1개로 감소되는 것이다. 적용할 수 있는 가중치 값은 노드수(N-1)를 요청하는 작업 프로세스 수로 나누었을 때의 결과 값((N-1)/P) 이상을 요구함을 가정한다. 또한 최소의 가중치 값이 만족되지 않을 경우, 각 노드당 하나의 프로세스를 생성하여 수행함을 가정한다.

식(5)는 제안한 방법의 수식을 나타내며 식(4)에 의해 결정된 노드별 프로세스 수에 가중치 값을 적용하여 각 노드에 생성되는 작업 프로세스 수를 줄임으로써 전체성능을 향상시킬 수 있다.

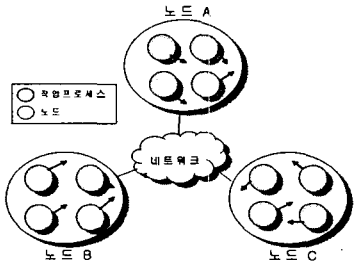


그림 2. 네트워크 성능을 고려하지 않고 노드별 균등 생성 방법
Fig 2. Uniformity creation method at each node without consideration of network performance

기존방법인 (그림 2)는 전체 노드 수를 N이라 할 때 하나의 노드는 작업에 참여하지 않고 요청 및 결과 값만을 확인하는 순수 클라이언트이므로 작업에 참여한 노드 수는 N-1이 된다. 사용자가 계산 작업을 위해 요청한 전체 작업 프로세스 수가 12개인 경우, 네트워크 성능을 고려하지 않고 N-1개의 노드 수만큼 균등하게 4개씩 노드별로 프로세스들을 생성한다. 하지만 계산 그리드 상에서 네트워크 의존적인 프로그램을 수행할 경우 전체 수행시간 중 상대적으로 네트워크상에서 소요되는 시간이 크기 때문에 이를 고려하지 않고 균등하게 노드별로 프로세스를 생성하면 프로세스 간 통신 시 지연시간, 대역폭의 영향을 받아 결국, 전체 수행시간의 단축을 기대할 수 없다.

(그림 3)은 식(4)에 의한 방법으로 참여한 노드 수가 3개이고 전체 작업 프로세스 수를 12개 요청했을 경우 네트워크 성능요소인 지연시간, 대역폭을 혼합하여 계산된 노드별 성능비율이 20%, 30%, 50% 일 때 2, 4, 6개 씩 작업 프로세스들이 노드별로 생성된다. 네트워크 성능이 우수한 노드에 보다 많은 프로세스들을 생성시켜 그 만큼의 네트워크 소요시간을 단축시킴으로써 성능을 향상시킬 수 있다. 하지만 균등방식과 같이 요청된 작업 프로세스 수(12개)가 동일한 상태에서 노드별 작업 프로세스들 간의 통신시간이 여전히 많기 때문에 전체 수행시간 중 네트워크 통신시간을 단축시키는 방법이 고려되어야 한다.

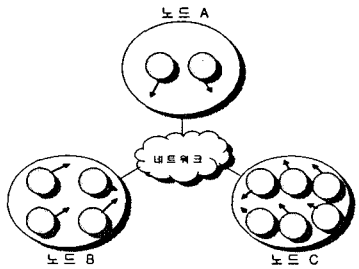


그림 3. 네트워크 성능정보를 고려한 방법
Fig 3. The method that considers network performance information

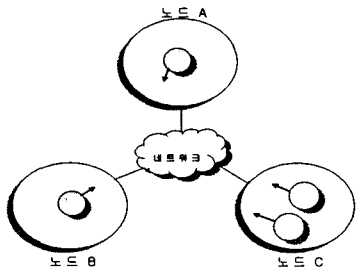


그림 4. 제안된 방법
Fig 4. The proposed method

노드 A : 2개 * 0.3 = 0.6(1개)
노드 B : 4개 * 0.3 = 1.2(1개)
노드 C : 6개 * 0.3 = 1.8(2개)

그림 5. 제안된 방법의 프로세스 수 계산 방법
Fig 5. Process number calculation method of the proposed method

그래서 본 논문에서 제안하는 (그림 4) 방법은 네트워크 의존적인 프로그램 경우에 전체 수행시간 중 네트워크의 소요시간이 노드의 계산 시간보다 많은 비중을 차지하기 때문에 전체 작업 프로세스 수를 감소시켜 네트워크의 소요시간을 줄임으로써 전체 수행시간을 단축시킬 수 있다. 가중치 값을 적용하여 노드별 작업 프로세스 수를 감소시킬 수 있으며 그 만큼의 네트워크 통신시간이 감소하게 되어 전체 수행시간을 향상시킬 수 있다.

(그림 5)는 (그림 3)에 의해 계산된 노드별 작업 프로세스 수 2, 4, 6개 각각에 가중치 30%를 적용한 예이다. 감소된 작업 프로세스 수에 따라 프로세스들로의 전달되는 작업량은 증가한다. 가령, 1부터 120까지의 데이터들을 기존 12개의 프로세스들은 균등하게 10개씩 전달받아 작업을 처리한다. 하지만 작업 프로세스 수가 가중치에 의해 4개로 감소되면 프로세스 별로 30개씩 전달받아 작업을 처리해야 한다. 이렇게 전달되는 작업량이 증가하기 때문에 네트워크 상으로 데이터 전송에 영향을 미치는 대역폭을 고려해야 한다. 따라서 본 논문에서의 네트워크 의존적인 프로그램을 수행할 때, 지연시간과 더불어 대역폭도 함께 작업 프로세스 수를 결정함에 있어 고려가 되기 때문에 그렇지 않은 기존방법에 비해 성능향상을 기대할 수 있다.

3.3 계산 의존적인 프로그램

계산 의존적인 프로그램은 전체 시간 중 네트워크를 통해 소요되는 시간보다 노드별로 수신된 작업량에 대한 처리시간이 보다 많은 비중을 차지하는 프로그램이다.

네트워크 의존적인 프로그램은 지연시간, 대역폭 상태에 따라 소요되는 시간이 전체 실행시간을 좌우하게 된다. 하지만 네트워크에서의 소요시간보다 노드별 계산시간이 상대적으로 많은 비중을 차지하는 프로그램에서는 노드별 사용가능한 cpu 자원에 따라 전체 수행시간이 결정된다. 따라서 본 논문에서는 네트워크에서 소요되는 시간보다 노드별 cpu 자원의 이용시간이 상대적으로 큰 응용프로그램을 고려하기 위해 전송되는 작업량과 노드에서 수행될 cpu 자원을 고려하여 작업 프로세스 수를 결정하는 방법을 제안한다. 다음은 계산 의존적인 프로그램 수행시 네트워크 상태정보와 노드별 cpu 자원을 고려하여 노드별 작업 프로세스 수를 결정하기 위해 사용된 용어 정의와 수식을 나타낸다.

- Ni_availablecpu : 노드 i의 사용가능한 cpu 성능 비율 값을 의미한다.
- total_availablecpu_sum: 전체 노드들의 사용가능한 cpu 성능비율 값의 총합을 의미한다.
- Ni_availablecpu_percentage: 노드 i의 사용가능한 cpu 성능비율 값을 의미한다.
- Ni_latency_bandwidth_availablecpu_sum: 노드 i의 지연시간, 대역폭, 사용가능한 cpu 자원에 대한 성능비율의 총합을 의미한다.
- Ni_latency_bandwidth_availablecpu_percentage: 노드 i의 지연시간, 대역폭, 사용가능한 cpu 자원에 대한 성능비율 값을 의미한다.
- Ni_calculation_process: 계산 의존적인 프로그램 수행시 노드 i에 생성될 작업 프로세스 수를 의미한다.

- Ni_latency_percentage = Ni_normal_value / total_latency_normal_sum ..... (식 6)

- Ni_bandwidth_percentage = Ni_bandwidth / total_bandwidth_sum ..... (식 7)

- Ni_availablecpu_percentage = Ni_availablecpu / total_availablecpu_sum ..... (식 8)

- Ni_latency_bandwidth_availablecpu_sum = Ni_latency_percentage+Ni_bandwidth_percentage +Ni_availablecpu_percentage

- Ni_latency_bandwidth_availablecpu_percentage = Ni_latency_bandwidth_availablecpu_sum / C ..... (식 9)

Ni_calculation_process = ROUND (P * Ni_latency_bandwidth_availablecpu_percentage) ..... (식 10)

네트워크 성능정보에 대한 혼합 성능비율은 3.2절과 같이 측정된 각 노드의 지연시간, 대역폭 값을 참여 노드의 지연시간, 대역폭의 총합으로 나누어 계산한다(식 (6), 식 (7)). 노드별 cpu 성능비율도 측정된 cpu 성능 값을 전체 노드의 사용가능 cpu 값의 총합으로 나누어 구한다(식 (8)). 노드별 네트워크 성능비율과 사용가능 cpu 비율을 혼합한 각 노드의 성능비율은 지연시간, 대역폭, cpu 성능비율의 총합에 대해서 혼합한 성능요소의 수인 C(=3)로 나누어 계산한다(식 (9)). 마지막으로, 계산된 3가지 성능요

소에 대한 성능비율 값을 사용자가 요청한 작업 프로세스 수에 적용하면 노드별 작업 프로세스 수가 결정된다(식 (10)).

지연시간, 대역폭만을 고려하게 되면 네트워크 소요시간만큼 전체 수행시간은 단축될 수 있지만 노드별 cpu 자원을 고려하지 않을 경우 전체 수행시간의 향상은 기대할 수 없게 된다. 그래서 계산 의존적인 프로그램은 네트워크 자원과 노드별 사용가능한 cpu 성능을 고려하여 작업 프로세스 수를 결정함으로써 성능을 향상시킬 수 있다. 그리고 작업 프로세스 수 관점에서는 작업을 처리하는 노드의 cpu 자원에 의존적이기 때문에 프로세스 수를 감소하기보다 사용자가 요청한 수만큼 이용하는 것이 전체 수행시간을 단축할 수 있다.

Ⅳ. 시스템 구조

본 논문에서는 노드 간 네트워크 상태정보 및 노드별 사용가능한 cpu 성능정보 수집을 위해서 NWS를 이용한다. 각 노드에는 자신의 상태정보를 일정시간 간격으로 수집하는 센서들이 작동하며 노드별 센서들은 수집한 정보들을 메모리 서버로 전송한다. 수집된 정보들은 메모리 서버에서 일관성 있게 관리되고 사용자들은 이를 이용한다. 하지만 NWS는 성능정보별 분석 및 분류할 수 있는 기능이 없다. 따라서 본 논문에서는 성능정보별로 노드들을 분류하여 추후에 노드별 성능비율 계산에 사용할 수 있게끔 라이브러리 형태로 구현하여 추가하였다.

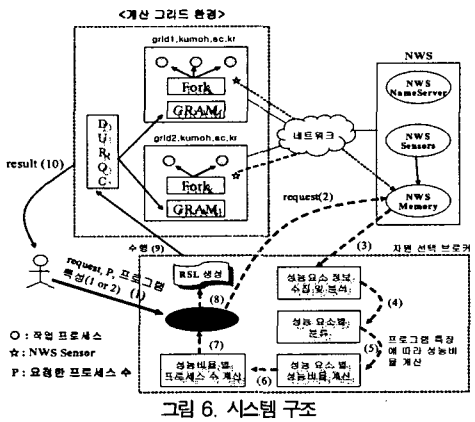


그림 6. 시스템 구조
Fig 6. System architecture

(그림 6)은 NWS로부터 수집한 정보를 기반으로 성능요소별 분석 및 분류를 통해 MPI 프로그램의 특성에 따라 노드별 성능비율을 계산한 후, 성능비율에 따른 작업 프로세스 수를 결정하여 최종적인 RSL 파일을 자동으로 만들어 수행하는 것을 보여준다.

기존 글로버스 툴 킷을 이용하는 사용자들은 자원의 위치정보, 네트워크 상태정보와 노드별 성능정보를 고려하여 작업들을 분배시켜 주는 브로커 기능이 제공되지 않고 있다. 그래서 노드별 작업 프로세스 수를 결정하기 위해서는 MDS를 통해 노드들의 일반적인 정보(cpu, memory, disk size) 만을 이용하여 각 노드에 균등하게 작업 프로세스 들을 생성하여 작업을 수행한다. 또한 노드들의 성능정보를 RSL 스크립트 파일에 반영하기 위해서는 사용자의 또 다른 노력이 필요하다.

```
"grid1.kumoh.ac.kr" 10
"grid2.kumoh.ac.kr" 10
```

(a). machines 파일 내용

```
mpirun -dumprsl -np 20 calculation > calculation.rsl
mpirun -globusrsl calculation.rsl
```

(b). RSL 스크립트 파일 생성 및 작업 수행

```
+
( &(resourceManagerContact="grid1.kumoh.ac.kr")
(count=10)
(label="subjob 0")
(environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
(LD_LIBRARY_PATH /usr/local/globus/lib/))
(directory="/home/shcho/Evaluation")
(executable="/home/shcho/Evaluation/calculate_20")
)
( &(resourceManagerContact="grid2.kumoh.ac.kr")
(count=10)
(label="subjob 10")
(environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
(LD_LIBRARY_PATH /usr/local/globus/lib/))
(directory="/home/shcho/Evaluation")
(executable="/home/shcho/Evaluation/calculate_20")
)
```

(c). MPICH-G2 명령어를 통해 생성된 RSL 파일
그림 7. 기존 글로버스 툴 킷 수행 방법
Fig 7. The existing globus toolkit execution method

글로버스를 통해 노드별로 수행할 작업내용을 담고 있는 RSL 파일을 작성하기 위해서는 (그림 7)과 같이 직접 모든 노드 자원의 위치정보 등을 조사하여 수동으로 RSL 스크립

트 문서를 작성하거나, MPICH-G2[16] 명령어를 통해 RSL 파일을 만들어 수행해야 하는 불편함이 있다. 물론 이럴 경우 네트워크 상태와 노드별 성능정보가 제대로 반영되지 않은 RSL 스크립트 파일을 작성하게 된다.

본 논문에서는 MPI 프로그램의 특성을 네트워크 의존적인 경우와 계산 의존적인 상황으로 분류하여 네트워크 상태 정보 및 노드별 사용가능한 cpu 성능정보를 적절히 반영시켜 자동으로 최적의 RSL 파일을 생성하는 자원 선택 브로커를 구현하였다. 사용자는 수행 요청과 함께 계산 작업을 위해 필요한 작업 프로세스 수만을 자원선택 브로커의 RSL-Maker에게 전달한다. 나머지 모든 단계들은 브로커에서 NWS로부터 수집된 정보들이 반영된 RSL 문서를 생성하여 계산 그리드 환경으로 전송한 후, 최종적인 결과 값을 사용자에게 전달한다.

```
Algorithm Make_Automatic_RSL(Node:작업에 참여한 노드)
{
 /* 노드별 latency 측정 */
 latency = Get_Latency();
 /* 노드별 bandwidth 측정 */
 bandwidth = Get_Bandwidth();
 /* 노드별 사용가능한 cpu 바율 측정 */
 availableCpu = Get_AvailableCpu();
 /* 측정된 정보 기록 */
 Write_Latency_Bandwidth_Cpu(latency, bandwidth, cpu);
 /* 측정된 latency 평균값 계산 */
 Average_of_Latency(latency, Node);
 /* 측정된 bandwidth 평균값 계산 */
 Average_of_Bandwidth(bandwidth, Node);
 /* 측정된 사용가능 cpu 바율 평균값 계산 */
 Average_of_Cpu(availableCpu, Node);
 /* 노드별 성능비율과 프로세스 수 계산 */
 Calculate_of_Process(Node);
 /* 네트워크 성능정보 및 cpu 자원이 반영된 RSL 파일생성*/
 Make_RSL(rsl, Node);
}
```

그림 8. 작업 프로세스 수의 결정과 RSL파일 생성 방법
Fig 8. Determination of the number of work process and RSL file creation method

(그림 8)은 프로그램 특성을 고려하여 네트워크 성능정보와 cpu 정보를 기반으로 노드별 작업 프로세스 수를 결정하는 방법을 나타낸다. 네트워크 의존적인 프로그램인 경우 NWS로부터 노드별 지연시간, 대역폭 값을 측정한 후 각 지연시간, 대역폭에 대한 평균값을 구한다. Calculation_of_Process() 함수에서는 프로그램 특성을 파악하여 필요한 구성요소(지연시간+대역폭)에 대한 성능비율 값을 계산하고 사용자가 요청한 작업 프로세스 수를 성능비율에 적용하여 (그림 9)와 같이 노드별 작업 프로세스 수를 결정한다. 또한, 프로세스 수의 감소를 위해 가중치 값을 적용하여 노드별 프로세스 수를 결정하는 부분도 이곳에서 이루어진다. 마지막 단계인 Make_RSL() 함수에서는 이전 단계의 프로세스 수를 기반으로 RSL 파일을 생성하여 최종적인 계산 작업을 수행한다.

계산 의존적인 프로그램인 경우에는 지연시간, 대역폭, cpu 혼합 정보를 기반으로 노드별 성능비율을 계산하여 노드별 작업 프로세스 수를 결정한다. 마지막 단계에서는, (그림 9)와 같이 결정된 작업 프로세스 수를 기반으로 RSL 문서를 자동으로 생성하여 실질적인 계산 작업을 수행한다. 또한 (그림 6)에서 2-8 단계 수행절차는 사용자 요청과 별개로 진행하여 네트워크 성능정보 수집 및 분류 등과 같이 불필요하게 소요되는 시간을 단축한다.

```
***** 초기 측정값 결과*****
latency average : 11.436800  26.850803  26.565601
bandwidth average : 11.303889  2.468230  11.401472
cpu average : 0.899158  0.267724  0.569874
생성할 프로세스 수: 12.000000

***** cpu에 따른 각 노드에 생성될 프로세스 수 *****
grid1.kumoh.ac.kr  6
grid3.kumoh.ac.kr  4
grid2.kumoh.ac.kr  2
***** latency+bandwidth 혼합한 각 노드 프로세스 수 *****
grid1.kumoh.ac.kr  5
grid2.kumoh.ac.kr  2
grid3.kumoh.ac.kr  5
**** latency+bandwidth+cpu 혼합한 각 노드 프로세스 수 ****
grid1.kumoh.ac.kr  6
grid2.kumoh.ac.kr  2
grid3.kumoh.ac.kr  4

RSL 파일이 생성되었습니다!!!
```

그림 9. 수행 결과
Fig 9. Execution result

V. 실험결과 및 분석

5.1 실험환경 및 평가방법

〈표 1〉은 성능평가를 위해 사용된 구성요소들을 보여준다. 실험에 참여한 노드들은 다양한 성능들로 구성하였으며 10~100Mbps 이더넷으로 연결된 5대의 노드들이며 성능평가를 위해 다양한 작업 프로세스 수, 노드 수, 작업량을 적용하여 수행하였다.

네트워크 성능정보 획득을 위해서는 NWS 사용하고 동적인 네트워크 상황을 구성하기 위해 네트워크 트래픽을 발생시키는 iperf를 이용하여 실험환경을 구성하였다. 계산 그리드 환경에서 수행되는 응용 프로그램의 특성을 시스템에서 임의로 판단하기는 어렵다. 그래서 본 논문에서는 사용자로부터의 입력을 통해 네트워크 의존적인 프로그램은 정수 값 1을, 계산 의존적인 프로그램은 정수 값 2를 입력받아 처리함을 가정하여 실험을 하였다.

표 1. 성능평가를 위해 사용된 구성요소
Table 1. Elements which is used for performance evaluation

항 목	내 용
운영체제	RedHat Linux 9.0 (커널버전 : 2.4.20-8)
소프트웨어	Globus Toolkit 2.2.2 MPICH-G2 NWS 2.8.1 iperf-1.7.0 (17)
참여노드 CPU/Memory	grid1.kumoh.ac.kr: P4 1.7GHz(768MB) grid2.kumoh.ac.kr: P3 666MHz(128MB) grid3.kumoh.ac.kr: P4 1.7GHz(256MB) grid4.kumoh.ac.kr: P4 1.7GHz(384MB) grid5.kumoh.ac.kr: P4 1.7GHz(128MB)

표 2. 성능평가의 실험방법
Table 2. Experimentation method of the performance evaluation

프로그램 특징	평가 방법
네트워크 의존적	1. 노드수(4), 작업량(512k)을 고정한 상태에서 작업 프로세스 수에 따른 실험 2. 노드수(4), 작업프로세스 수(36)를 고정한 상태에서 전송 작업량에 따른 실험 3. 작업 프로세스 수와 작업량을 노드 수에 따른 실험
계산 의존적	1. 노드수(4), 작업량(500x500)을 고정한 상태에서 작업 프로세스 수에 따른 실험 2. 작업 프로세스 수(24), 작업량(1M)을 노드 수에 따른 실험

〈표 2〉는 성능평가 방법에 대해서 언급하고 있으며 계산 그리드 환경에서 실행되는 프로그램의 특성을 네트워크 의존적인 프로그램과 계산 의존적인 프로그램으로 나누어 평가한다.

네트워크 의존적인 프로그램의 성능평가를 위한 계산 방법은 마스터에 생성된 마스터 프로세스에서 슬레이브 노드에 생성된 작업 프로세스들로 처리할 작업을 전송한다. 각 노드의 작업 프로세스들은 수신된 작업 데이터들을 누적 계산을 한 후 그 결과 값을 마스터에 전송한다. 마스터에서는 작업 프로세스 들이 계산한 처리 결과 값들을 취합하여 사용자에게 알려준다. 성능평가 방법은 노드수와 작업량을 고정한 상태에서 작업 프로세스 수를 증가했을 때 어떤 변화가 일어나는지 살펴보고, 노드수와 작업 프로세스 수가 고정된 상태에서 전송되는 작업량을 증가 시켰을 때, 균등방식인 기존방법과의 성능비교를 한다. 마지막으로 노드수의 변화에 따른 성능평가를 실험을 통해 확인한다.

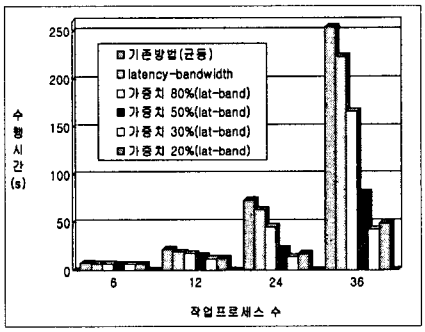
계산 의존적인 프로그램의 성능평가를 위한 계산 방법은 성능평가에 있어 가장 일반적으로 사용되는 방법 중에 하나인 행렬 곱셈식을 이용하였다. 노드별 cpu 의존적인 프로그램을 만들기 위해 슬레이브 노드의 작업 프로세스들에게 500x500 행렬이 저장된 파일을 읽어 계산하게 하였으며 처리한 결과 값은 마스터로 전송한다. 마스터에서는 그 결과 값들을 취합하여 사용자에게 알려준다.

노드별 계산 의존적인 프로그램에 대한 성능평가 방법은 작업 프로세스 수와 노드 수에 따른 결과를 실험을 통해 확인한다. 노드 수에 따른 실험은 3대에서 5대까지 확장하여 실험하였다. 참여하는 노드수의 증가에 따라 시스템에 끼치는 영향을 최소화하기 위해 NWS, 글로버스, 자원 선택 브로커를 독립적으로 구성하여 동작되기 때문에 노드 수의 확대로 인한 영향은 문제가 되지 않는다.

5.2 실험결과 및 분석

5.2.1 네트워크 의존적인 프로그램에 따른 결과

(그림 10)은 동적인 환경에서 노드 수 4개, 작업량을 512k로 고정하여 사용자가 요청한 작업 프로세스 수를 증가하여 수행한 결과이다. 작업 프로세스 수가 증가할수록 수행시간이 단축되지 않고 오히려 증가함을 알 수 있다. MPI 기반 프로그램은 네트워크를 통해 작업량과 결과 값이 송수신 되므로 노드별 계산시간보다 네트워크 상에서의 소요시간이 많은 비중을 차지하기 때문에 작업 프로세스 수가 증가하면 전체 성능이 저하된다. 그래서 네트워크 의존적인 프로그램을 수행하여 전체 성능을 향상시키기 위해서는 네트워크 통신시간의 비중을 줄이는 방법, 즉 작업 프로세스 수를 감소시켜 네트워크 상에서의 통신시간을 줄여야 한다.

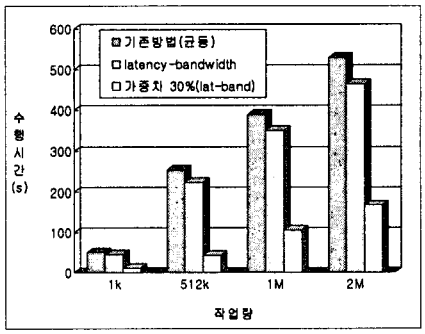


노드수 : 4개, 작업량 : 512k

그림 10. 작업 프로세스 수에 따른 결과
Fig 10. Result by the number of work process

(그림 10)에서와 같이 노드 수가 4개, 작업량이 512k, 네트워크가 동적인 상태에서 작업 프로세스 수가 증가할수록 전체 프로세스 수를 결정하는 가중치 30%에 대한 성능이 기존방법(균등)과 지연시간-대역폭보다 595%, 524% 성능이 향상됨을 알 수 있다. 이는 가중치 30%에 대한 작업 프로세스 수가 전체 수행시간 중 네트워크 수행시간이 가장 작게 소요되었기 때문에 수행시간이 단축된 것이다. 하지만 가중치가 20% 인 경우 성능이 저조한 이유는 작업 프로세스 수를 감소한 만큼 프로세스 별 수행해야 할 작업량이 증가하여 그 만큼의 실행시간이 전체 수행시간에 반영되었기 때문이다.

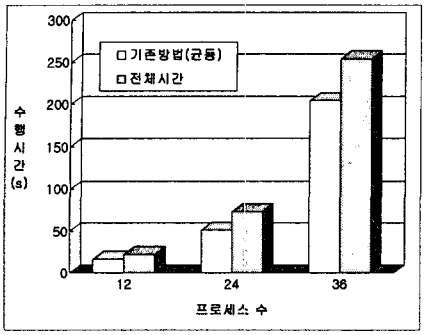
(그림 11)은 동일한 노드 수와 작업 프로세스 수에서 네트워크를 통해 전달되는 작업량에 따른 수행결과를 나타낸다. 작업량의 크기가 증가할수록 제안한 방법이 기존방법(균등)과 지연시간-대역폭 보다 316%, 278% 성능이 향상됨을 알 수 있다. 네트워크 의존적인 프로그램은 상대적으로 노드별 프로세스들의 계산시간은 일정하다. 그래서 기존 방법들은 작업 프로세스 수의 변화 없이 네트워크를 통해 전달되는 작업량이 커지면 네트워크 상태를 고려한 만큼의 성능향상은 기대할 수 있지만 더 이상의 전체 수행시간은 단축되지 않는다.



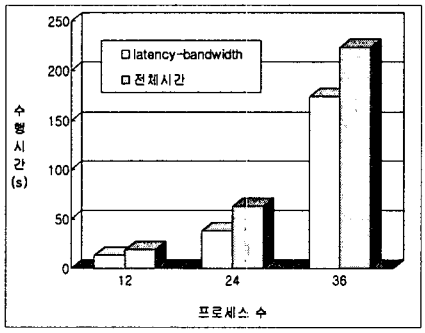
노드수 : 4개, 작업 프로세스 수 : 36개

그림 11. 작업량에 따른 결과
Fig 11. Result by quantity of work

(그림 12)는 작업 프로세스 수의 증가에 따른 전체 수행시간 중 네트워크 상에서 소요되는 시간을 보여주고 있다. 결과에서 알 수 있듯이 네트워크 의존적인 프로그램은 전체 수행시간 중 네트워크 상에서 소요되는 비율이 기존방법(균등)은 80%, 지연시간-대역폭 방식은 78%, 제안방법은 56% 임을 알 수 있다. 그래서 MPI 기반 네트워크 의존적인 프로그램을 계산 그리드 상에서 수행 할 경우 노드별 작업 프로세스들의 계산시간보다 네트워크 상에서 소요되는 시간을 줄여야 함을 보여준다. 네트워크의 수행시간을 단축하기 위해서는 지연시간, 대역폭을 고려하고 전체 작업 프로세스 수를 줄여 프로세스 간 통신시간을 감소시켜야 한다.



(a) 균등방식의 네트워크상에서 소요된 시간

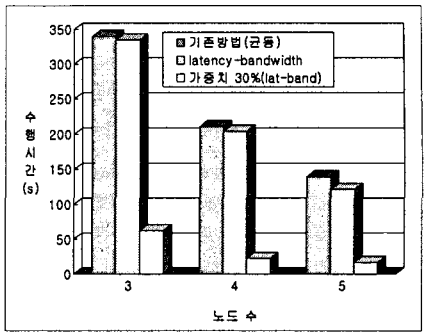


(b) 지연시간-대역폭 방식의 네트워크상에서 소요된 시간

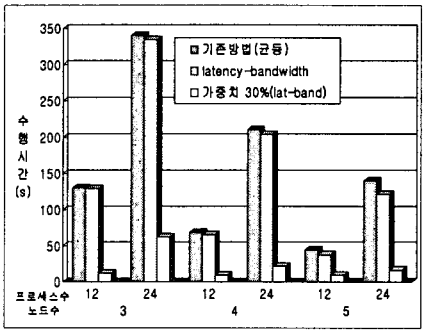
그림 12. 전체 수행시간 중 네트워크에서의 소요시간
Fig 12. Networking time in the total execution time

(그림 13)은 노드 수를 증가하였을 때 작업 프로세스 수와 작업량에 대한 실험결과를 보여준다. 작업 프로세스 수가 24개이고 작업량이 1M인 상황에서 노드수를 증가 하였을 때 제안한 방법이 기존방법(균등), 지연시간-대역폭방식 보다 954%, 924% 성능이 향상됨을 알 수 있다. 기존 방법과 제안방법에서 노드 수가 증가할수록 성능이 향상되는 것은 사용가능한 지연시간, 대역폭이 노드 수가 작을 때보다 많아지면 네트워크 자원의 사용 집중 현상을 다른 노드로 분산시켜 네트워크에서 소요되는 시간이 단축되었기 때문이다.

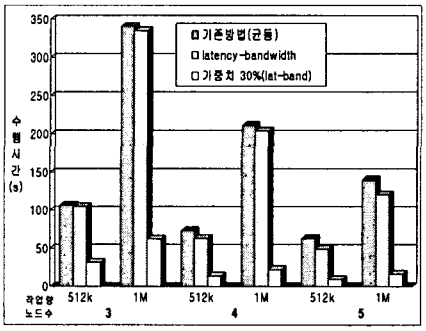
또한 작업 프로세스 수의 증가에 따른 노드 수 증가 시 제안된 방법의 성능이 향상되었고, 작업량 증가에 따른 실험에서도 노드 수가 증가할수록 기존방법보다 성능이 우수하였다. 그리고 작업량, 작업 프로세스 수가 클수록 제안방법이 기존방법보다 성능이 큰 폭으로 향상됨을 확인 할 수 있다. 이는 작업 프로세스 수와 작업량이 커지면 그 만큼의 네트워크 통신시간이 길어져서 전체 수행시간이 증가하기 때문이다.



(a) 작업 프로세스 수 : 24개,
작업량 : 1M



(b) 작업량 : 1M



(c) 작업 프로세스 수 : 24개

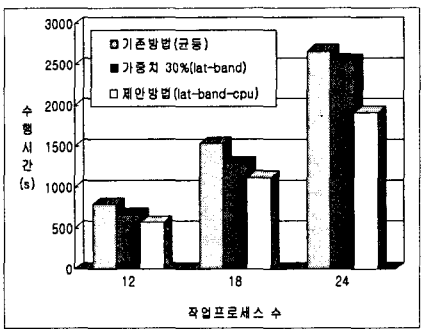
그림 13. 노드 수에 따른 결과
Fig 13. Result by the number of node

따라서 네트워크 의존적인 프로그램에서 노드 수, 작업량의 변화에 따른 작업 프로세스 수를 결정함에 있어 전체 수행시간 중 네트워크 통신시간이 노드별 프로세스들의 계산시간보다 많은 부분을 차지하기 때문에 지연시간, 대역폭을 고려하여 노드별로 생성되는 작업 프로세스 수를 줄임으로써 전체 성능이 향상됨을 실험을 통해 확인할 수 있다. 본 논문에서 실험을 통해 얻어진 가중치 30%에 대한 평가 값

이 가장 좋은 성능을 보였지만 네트워크 상태, 노드 수, 프로그램의 특성, 작업 프로세스 수 등에 따라 유동적으로 그 최적의 값이 달라질 수 있다.

5.2.2 계산 의존적인 프로그램에 따른 결과

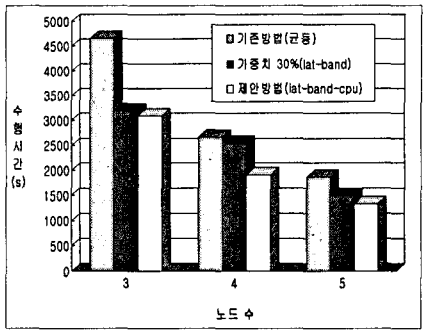
(그림 14)는 MPI 기반 노드별 계산 의존적인 프로그램에 대해서 작업 프로세스 수의 증가에 따른 실험결과를 보여준다. 계산 의존적인 프로그램을 네트워크 자원에 국한시킨 경우 보다 노드별 사용가능한 cpu 자원을 함께 고려하여 작업 프로세스 수를 결정했을 때 성능이 우수함을 보여준다. 네트워크 자원만을 고려한 가중치 30%에 대한 결과에서 성능향상이 저조함을 나타내고 있다. 이것은 계산 의존적인 프로그램에서는 네트워크 상에서 소요되는 시간 보다 노드별 프로세스들의 처리시간이 많은 비중을 차지하기 때문에 작업 프로세스 수의 감소와 프로세스 별 처리할 작업량의 증가에 따라 수행시간이 그만큼 길어졌기 때문이다.



노드수 : 4개, 작업량 : 500*500

그림 14. 작업 프로세스 수에 따른 결과
Fig 14. Result by the number of work process

(그림 15)는 고정된 작업량과 작업 프로세스 수 환경에서 노드 수의 증가에 따른 실험결과를 나타낸다. 모든 경우 노드 수가 증가할수록 전체 수행시간이 단축됨을 보여주고 있다. 이는 노드 수가 증가할수록 네트워크 자원의 사용이 집중되는 것을 막아주고 계산 작업에 사용되는 보다 우수한 cpu 자원 이용이 그 만큼 증가하여 전체 수행시간에 반영되었기 때문이다.



작업 프로세스 수 : 24개,
작업량 : 500*500

그림 15. 노드 수에 따른 결과
Fig 15. Result by the number of node

따라서 계산 그리드 환경에서 계산 의존적인 프로그램을 실행 할 경우에는 전체 수행시간 중 네트워크 상에서의 소요시간보다 노드별 프로세스들의 작업 처리 시간이 많은 부분 반영되기 때문에 작업 프로세스 수를 증가하여 cpu 사용량을 높여 계산시간을 줄일 수 있으며 또한 노드 수를 증가하여 네트워크 소요시간을 감소시켜 전체 수행시간을 단축시킬 수 있음을 실험을 통해 확인할 수 있다.

VI. 결론 및 향후연구

계산 그리드 환경은 LAN/WAN으로 구성되어 네트워크를 통해 작업 및 결과 값이 송수신되며 수신된 작업량을 노드별 성능에 따라 전체 수행시간이 결정되기 때문에 계산 작업 시 반드시 고려되어야 한다. 본 논문에서는 수행되는 응용 프로그램의 특성을 네트워크 의존적인 프로그램과 노드별 계산 의존적인 프로그램으로 구분하여 각 특성에 맞게끔 네트워크 성능정보와 cpu 정보를 이용하여 노드별 성능비율을 계산하고 자동으로 RSL 파일을 생성하여 작업을 수행하는 자원 선택 브로커를 제안한다.

실험결과, 네트워크 의존적인 프로그램 관점에서 네트워크 정보를 고려하지 않은 균등방식과 네트워크 상태만을 고려한 방법보다 네트워크 상태정보를 고려한 후, 가중치 30%를 적용했을 경우 프로세스 수가 증가할수록 597%, 524% 성능이 향상되었다. 전송되는 작업량과의 관계에서도

제안된 방법이 기존 방법보다 316%, 278% 성능향상을 보였다. 노드 수 증가에 따른 실험에서도 작업량과 수행되는 작업 프로세스 수가 증가 할수록 기존방법보다 954%, 924% 성능이 향상되었다. 계산 의존적인 프로그램에 대한 실험에서는 네트워크 상태정보 뿐만 아니라 실질적인 계산 작업이 이뤄지는 노드들의 성능정보를 함께 고려하여 반영시킨 결과, 노드별 cpu 성능정보를 고려하지 않은 방법보다 작업 프로세스 수와 노드수를 증가했을 때 성능이 향상됨을 알 수 있었다.

따라서 계산 그리드 환경에서 수행되는 프로그램의 특성을 고려하여 적용되는 성능정보에 따라 성능향상 폭이 커다는 것을 알 수 있다. 특히 네트워크 의존적인 프로그램에서는 단지 네트워크 상태정보만을 고려하여 계산 작업을 수행하면 더 이상의 성능향상이 이뤄지지 않음을 알 수 있다. 그래서 네트워크 의존적인 프로그램의 특성상 노드별 계산 시간은 일정하기 때문에 수행시간을 단축시킬 수 있는 방법은 전체 프로세스 수를 줄여 네트워크에서 소요되는 시간을 줄임으로써 전체 성능이 향상됨을 실험을 통해 확인하였다. 그리고 노드별 계산 의존적인 프로그램 경우에는 네트워크 상태정보 뿐만 아니라 각 노드들의 사용가능한 cpu 비율을 계산 작업에 반영시켜야 한다. 노드별 계산 시간이 전체 수행시간 중 많은 비중을 차지하기 때문에 참여하는 작업 프로세스 수와 노드수를 증가할수록 성능이 향상됨을 알 수 있다.

향후 연구에서는 MPI 라이브러리의 성능 개선과 네트워크 상태정보와 노드별 cpu 성능정보 수집을 정확하게 수집/관리하여 계산 작업에 반영시키는 문제들을 추후 연구에서 고려되어야 할 사항이다. 그리고 본 논문에서는 프로그램 특성을 사용자의 입력을 통해 크게 네트워크 의존적, 계산 의존적으로 나누어 살펴보았다. 계산 그리드 상에서 수행되는 프로그램의 특성을 파악하는 것은 작업 프로세스 수를 결정하기 위해 반드시 선행되어야 한다. 하지만 사용자가 작성한 프로그램의 특성을 파악하는 것은 쉽지 않기 때문에, 향후 연구에서는 프로그램의 특성을 일반화하여 파악할 수 있는 방법이 연구되어야 할 것이다.

참고문헌

[1] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," International Journal of Supercomputer Applications, Vol. 11, No. 2, pp. 115-128, 1997.

[2] 이용주, "시멘틱 e-워크플로우 프로세스를 이용한 동적 웹 서비스 조합," 한국컴퓨터정보학회논문지, 제10권 제1호, 2005.

[3] 황선태, 심규호, "계산 그리드에서 워크플로우 기반의 사용자 환경 설계 및 구현," 한국컴퓨터정보학회논문지, 제10권 제4호, pp. 165-171, 2005.

[4] Network Weather Service, http://nws.cs.ucsb.edu

[5] RSL, http://www-fp.globus.org/gram/rsl_spec1.html.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, "Grid Information Services for Distributed Resource Sharing," Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[7] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing," Software Practice and Experience Journal, Vol. 32, No. 2, pp. 135-164, Feb. 2002.

[8] C. Liu, L. Yang, I. Foster and D. Angulo, "Design and Evaluation of a Resource Selection Framework for Grid Application," Proceedings of IEEE International Symposium on High Performance Distributed Computing, pp. 63-72, 2002.

[9] J. Subhlok, P. Lieu, and B. Lowekamp, "Automatic Node Selection for High Performance Applications on Networks," Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel and Programming, pp. 163-172, 1999.

[10] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," Cluster Computing Vol. 5, No. 3, pp. 237-246, 2002.

[11] D. Abramson, R. Buyya, and J. Giddy, "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker," Future Generation Computer Systems (FGCS) Journal, Vol. 18, Issue 8, pp. 1061-1074, The Netherlands, October, 2002.

[12] K. L. Park, H. J. Lee, Y. J. Lee, O. Y. Kwon, S. Y. Park, H. W. Park, and S. D. Kim, "An Efficient Collective Communication Method for Grid Scale Networks," International Conference on Computational Science, pp. 819-828, 2003.

[13] N. Karonis, B. de Supinski, I. Foster, W. Gropp, E. Lusk and J. Bresnahan, "Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance," Processings of the 14th International Parallel and Distributed Processing Symposium(IPDPS '00), pp. 377-384, 2000.

[14] R. Wolski, N. T. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," Journal of Future Generation Computing Systems, Vol. 15, No. 5-6, pp. 757 - 768, October, 1999.

[15] 조수현, 김영학, "계산 그리드 상에서 각 노드의 작업 프로세스 수를 결정하기 위한 효율적인 방법," 한국콘텐츠학회논문지, 제5권, 제1호, pp. 189-199, 2005.

[16] I. Foster and N. Karonis, "A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems," Proceedings of Supercomputing 98, 1998.

[17] iperf-1.7.0, http://dast.nlanr.net/Projects/iperf/

저 자 소 개

조 수 현

2000년 2월 금오공과대학교 컴퓨터공학과(공학사)
2002년 2월 금오공과대학교 컴퓨터공학과(공학석사)
2006년 2월 금오공과대학교 컴퓨터공학과(공학박사)
2002년~2006년 2월 금오공과대학교 컴퓨터공학과 시간강사
2006년 3월 ~ 현재 : 금오공과대학교 컴퓨터공학부 계약교수
〈관심분야〉 Grid Computing, Embedded System

김 영 학

1984년 2월 금오공과대학교 전자공학과(공학사)
1989년 2월 서강대학교 전자계산학과(공학석사)
1997년 2월 서강대학교 전자계산학과(공학박사)
1989년 ~ 1997년 해군사관학교 전산과학과 교수
1998년 ~ 1999년 여수대학교 멀티미디어학부 교수
1999년 ~ 현재 : 금오공과대학교 컴퓨터공학부 부교수
〈관심분야〉 병렬 알고리즘, 분산 및 병렬처리, Embedded System