韓國IT서비스學會誌
第7卷 第4號
2008年 12月

LAN상의 비전용 노드를 포함한 HPC 클러스터의 확장에 의한 성능 향상

박 필 성*

Improving Performance of HPC Clusters by Including Non-Dedicated Nodes on a LAN

Pil-Seong Park*

■ Abstract ■

Recently the number of Internet firms providing useful information like weather forecast data is growing. However most of such information is not prepared in accordance with customers' demand, resulting in relatively low customer satisfaction. To upgrade the service quality, it is recommended to devise a system for customers to get involved in the process of service production, which normally requires a huge investment on supporting computer systems like clusters. In this paper, as a way to cut down the budget for computer systems but to improve the performance, we extend the HPC cluster system to include other Internet servers working independently on the same LAN, to make use of their idle times. We also deal with some issues resulting from the extension, like the security problem and a possible deadlock caused by overload on some non-dedicated nodes. At the end, we apply the technique in the solution of some 2D grid problem.

Keyword : Extension of an HPC cluster, Virtualization, Security, Avoiding a deadlock

논문투고일 : 2008년 07월 18일 논문수정완료일 : 2008년 09월 07일 논문게재확정일 : 2008년 09월 10일

* 수원대학교 컴퓨터학과

1. 서 론

최근 인터넷을 통하여 고객이 원하는 정보를 수집 가공하여 판매하는 업체들이 많이 생겨났다. 일례로 케이웨더(주), (주)첨성대 같은 업체들은 기상청의 예보자료를 웹, 음성 및 문자서비스 등을 통해 제공하고 있다. 그러나 일부 유료 서비스에 한해 약간의 가공을 거칠 뿐, 대부분 기상청의 자료를 그대로 전달해 줄 뿐이며, 고객이 원하리라 예상되는 정보를 미리 생성하여 제공하는 실정이다.

그러나 서비스사이언스 특히 IT서비스의 관점에서, 서비스 지배논리를 근간으로 고객이 서비스 생산과정에 공동생산자로 참여함으로써 서비스의 질과 생산성을 획기적으로 높일 수 있을 것이다[3]. 즉 고객은 일방적으로 제공하는 서비스를 이용만 할 것이 아니라, 자신의 요구를 반영하여 만족스런 서비스가 생산될 수 있는 체제가 필요하다.

이를 위해 대고객 웹 서비스에 투입되는 컴퓨터는 물론, 고객이 원하는 정보(예를 들어, 자기 동네의 특정 시점의 기상예보 등)를 최단시간 내에 생산할 고성능의 컴퓨터 시스템을 필요로 한다. 특히 기상 예측과 같은 시뮬레이션에는 격자를 사용하여 병렬연산을 수행하는 것이 일반적인데, 1990년대 이후 전용의 HPC 클러스터(High Performance Computing cluster)가 널리 사용되고 있다.

강력한 컴퓨터 자원을 투입하면 더 나은 서비스를 제공할 수 있으나 무조건 투자를 늘릴 수도 없는 형편이다. 다행히 추가적 예산 없이 LAN상의 컴퓨터들로 클러스터를 구성하여 활용할 수도 있다[1, 10, 16]. 그러나 이는 전용 클러스터와는 달리, 보안에 취약하며 계산 도중 비전용 노드의 과부하는 전체 연산 속도를 떨어뜨리며, 장애라도 발생하면 클러스터 전체가 교착상태에 빠져 연산 실패로 끝나는 등 해결해야 할 문제가 많다.

본 논문에서는 야간에만 활용 가능한 실습실 PC로 구성되는 [1]과는 달리, 소수의 전용 계산 노드로 구성된 클러스터를 확장하여 LAN상에서 본연의 임무를 수행하는 노드들의 유휴시간을 시간제약없이 활용하여 클러스터를 확장 구성하되, 앞에서 언급한 문제점들을 해결하는 일례를 제안한다. 이는 최근 차세대 IT 전략기술로서 각광받는 가상화의 한 형태로 볼 수도 있다. 그러나 이미 검증된 MPI 표준 함수만을 사용하여 응용 프로그램 수준에서 구현함으로써 이식성을 높이고, 2차원의 격자 문제의 연산에 적용하였다.

2. 관련 연구

2.1 가상화(Virtualization)

급변하는 IT 인프라의 활용에 있어 물리적 자원의 확장만으로는 한계에 부딪힌 상황에 있다. 시장조사 회사인 IDC(International Data Corporation)에 따르면 x86 서버의 평균 활용도는 총 용량의 15~20%에 불과하며[18], 기존 자원의 활용을 극대화하는 새 기술로 떠오른 것이 가상화 기술이다.

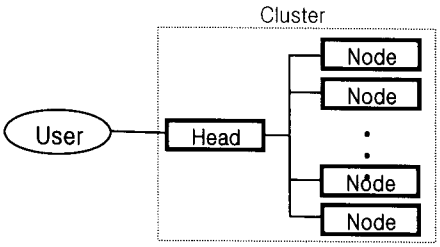
가상화란 하나의 물리적 자원을 논리적으로 분할해 사용하거나, 물리적으로 다른 여러 자원을 논리적으로 통합하는 기술로, 하드웨어부터 서비스에 이르기까지 여러 계층에 걸쳐 적용가능하다. 가상화는 1960년대 후반 가상 메모리에서 시작되었으나 극히 한정적으로 사용되어 왔을 뿐이다[2].

시장조사기관인 Gartner의 2003년 예측대로 가상화 시장은 연 60%의 성장세를 보이며 VMware, IBM, Microsoft 등의 소프트웨어 업체와 Intel, AMD 등의 프로세서 공급업체, Xen 같은 오픈소스 소프트웨어의 등장으로 많은 기업들이 시장에 진입하였다. 한편 최근 Gartner 그룹은 2009년을 선도할 10대 전략기술과 이에 대한 트렌드를 발표하였는데, 10가지 전략적 기술 분야중 가상화가 IT 기술전략 1순위로 나타났다[7]. 이런 기술들로 말미암아 기업들이 새로운 비즈니스 가치를 창출하고 탐색하며, 다양한 비즈니스 서비스와 솔루션을 새로이 개발할 수 있을 것이라고 예측되고 있다.

가상화 기술의 핵심은 서버 가상화이며, 점차 스토리지와 고객장치의 가상화, 호스티드 가상 이미지(hosted virtual image)로 확장될 것으로 전망되고 있다.

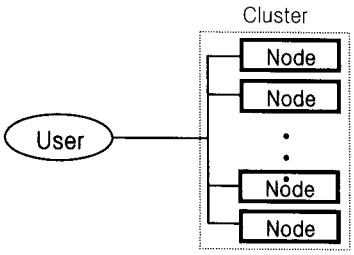
2.2 HPC 클러스터와 MPI

HPC 클러스터는 헤드(head) 역할을 하는 노드 아래 실제 작업을 수행할 계산 노드들로 구성된 비대칭 구조의 전용 클러스터([그림 1])가 일반적이다. 계산 노드는 외부에 노출되지 않아 보안상 유리하고, 다른 부하가 걸리지 않아 안정적이며 고정적 부하분산 정책을 사용할 수 있다는 장점이 있다.



[그림 1] 비대칭 클러스터

반면 개방된 LAN상에서 각자 역할을 수행하는 컴퓨터들의 유휴시간을 활용하는 대칭 구조의 클러스터로는 1990년대 중반 버클리 대학의 NOW (Network of Workstations)가 그 시초이다 [14]. 그러나 모든 노드가 외부에 노출되어 보안상 취약하며 관리가 어렵고, 부하 예측이 불가능하여 동적 부하분산이 주요 이슈가 된다([12] 참고).



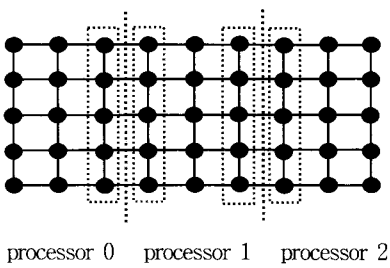
[그림 2] 대칭 클러스터

HPC 클러스터를 위한 미들웨어로는 메시지 패싱 방식 표준인 MPI(Message Passing Interface) [15]가 있으며, 이 표준에 따라 만들어진 MPI 라이브러리는 각 벤더들의 상용 라이브러리와 LAM, MPICH, OpenMPI 등 여러 종류가 있다.

그러나 MPI는 연산 실패를 극복할 방법의 표준을 제공하지 않아 여러 비표준의 결함 내성형 라이브러리가 개발되었다. 이들은 체크포인트/롤백(checkpoint/rollback)을 사용하는 것(MPI-CH V, Co-Check MPI, LA-MPI 등)과 메시지나 프로세스의 복제를 사용하는(MPI-FT 등) 두 종류로 나눌 수 있다[6, 11]. 이 외에 BLCR(Berkeley Lab Checkpoint/Restart)[13]과 같은 오픈 소스 체크포인팅 유틸리티가 있으나 적용가능한 운영체제 및 MPI는 극히 제한적이다. 따라서 이들의 사용은 프로그램의 이식성을 떨어뜨리게 된다.

2.3 격자 문제 및 동기 병렬 연산

기상예보를 포함한 자연과학 분야에서는 FDM (finite difference method)같은 기법을 사용하여 격자점에서의 값을 구하는 수치모델을 많이 사용한다. 예를 들어, [그림 3]은 영역 분할하여 3개의 프로세서로 계산하는 경우, 각 프로세서가 계산할 격자점들을 나타낸다. 다른 프로세서가 담당하는 영역과의 경계치(점선으로 된 사각형 내의 격자점의 값)는 인접 프로세서의 연산에도 필요하므로, 이 격자점들의 값이 갱신될 때마다 서로 교환해야 하며, 이런 연산을 수행하는 동기 병렬 알고리듬의 일반적 구조는 Algorithm 1과 같다.

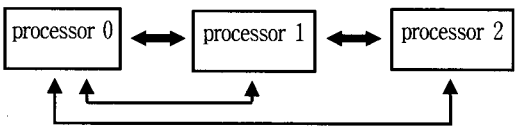


[그림 3] 3개의 프로세서를 사용한 영역분할

Algorithm 1 (각 프로세서)
만족스러울 때까지 다음을 반복한다.
1) 각자 격자점의 값을 업데이트한다.
2) 좌우측 프로세서와 경계치를 교환한다.
3) 수렴 판단에 필요한 정보를 계산하여 processor 0에게 보낸다.
4) processor 0은 정보를 종합하여 수렴여부를 판단하고, 연산의 계속 여부를 다른 프로세서들에게 알린다.

매 단계에서 동기화되므로 한 프로세서에 과부하가 걸리면 과부하가 해소될 때까지 모든 프로세서는 대기하거나 교착 상태에 빠지기도 한다.

이 알고리즘을 수행하는 프로세서들은 [그림 4]와 같이 나열되어 있다고 생각할 수 있고, processor 0이 전체적인 컨트롤을 담당한다면 프로세서들 간의 경계치 교환(Algorithm 1의 2))은 굵은 화살표를 통해, 전체의 컨트롤 관련 정보 교환(Algorithm 1의 3) 및 4))은 가는 화살표를 통해 이루어진다.



[그림 4] 프로세서의 배열 및 메시지 패싱

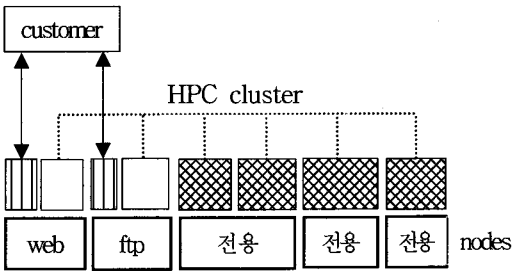
지금까지는 프로세서의 관점에서 서술하였으나, MPI에서 계산의 주체는 프로세스이므로 이하 프로세스를 기준으로 설명하기로 한다.

3. 클러스터의 확장 및 보안 강화

3.1 HPC 클러스터의 확장

[1]처럼 대부분의 대칭형 클러스터 연구는, 야간 및 주말의 유휴시간을 이용하는 것으로 한정되어 있다. 그러나 본 논문은, 전용의 HPC 클러스터를 LAN상의 모든 컴퓨터를 포함하여 확장하되 시간 제약 없이 활용하는 것이 목적이다.

예를 들어, [그림 5]는 3대의 계산 전용 노드(그 중 첫 번째 것은 2개의 CPU를 가짐을 가정)로 구성된 HPC 클러스터에, 대고객 서비스를 수행하는 웹 및 ftp 서버를 포함하여 클러스터를 확장한 것을 나타낸다. 제일 아랫단은 각 노드를 나타내며, 바로 위는 각 노드에서 실행중인 주요 프로세스를 나타낸다. ▩는 계산전용 MPI 프로세스, ▥는 웹 혹은 ftp 서버 프로세스, 그리고 □는 웹 및 ftp 서버 노드의 유휴자원을 활용하여 실행하는 MPI 프로세스를 나타낸다. 단, 웹 및 ftp 서버는 원래의 임무 수행을 방해받지 않고 과부하가 걸리는 것을 피하기 위해, 웹 및 ftp 서버 프로세스는 Linux의 nice 혹은 renice 명령으로 프로세스의 우선순위를 최대한으로 높임을 가정한다.



[그림 5] HPC 클러스터의 확장

3.2 확장된 클러스터의 구성요소 및 취약점

확장된 클러스터는 노드들이 외부에 노출되므로 다음과 같은 보안상의 취약점을 가진다.

- 클러스터의 노드들 전부 혹은 일부는 외부에 직접 노출되어 있다.
- 병렬 연산시, 노드간에 암호 없이 명령의 전달 실행이 가능해야 하는데, 보안에 취약한 "Berkeley R" 명령들(rlogin, rsh, rexec)이 주로 사용된다.
- 모든 노드는 보안에 취약한 NFS (network file system)를 통해 파일을 공유한다.

Berkeley R 명령과 NFS의 문제는 비대칭 클러스터의 경우에도 마찬가지이나, 노드가 노출되어 있지 않으므로 문제가 되지 않는다.

3.3 보안 강화 방안

암호화를 하지 않는 Berkeley R 명령 대신 ssh (secure shell)를 사용하기로 한다. ssh는 모든 메시지를 암호화하므로 전송에 시간이 더 걸릴 것 같으나 데이터를 압축해서 보내므로 오히려 전송 효율이 더 좋은 것으로 널리 알려져 있다.

또한 다음과 같이, 암호 없이 사용가능한 ssh의 인증키를 만들어 ssh를 통한 명령의 전달 실행이 가능토록 한다(RSA 키의 예를 든다, [19] 참고).

1) $ ssh-keygen-t rsa
   이 때 암호를 설정치 않아야 한다.
2) ~/.ssh/ 디렉토리의 공개키 id_rsa.pub을 authorized_keys라는 이름으로 복사한다.

원래 인증키의 사용을 위해서는 원격 서버로 공개키를 복사해야 하나, HPC 클러스터의 모든 노드는 NFS로 디렉토리를 공유하므로 2)와 같이 같은 디렉토리에 다른 이름으로 복사만 하면 된다.

한편 노드들은 외부에 노출된 상태에서 NFS로 파일을 공유하므로 각 노드에 방화벽을 설치하되 필수 포트만 열어준다. 문제는, NFS 서비스에 필요한 포트 중 유동적인 일부 포트는 고정하여야 방화벽을 사용할 수 있다는 점이다. NFS 작동에 필수적인 것은 portmapper(111), rpc.nfsd(2049), rpc.mountd, rpc.lockd, rpc.statd, rpc.quotad이다. 괄호 속의 숫자는 고정된 포트 번호이며, 번호가 지정되지 않은 것은 유동적이다. 유동적인 포트는 NFS의 설정 파일 /etc/sysconfig/nfs에 포트 번호를 지정함으로써 고정할 수 있다[17].

또한 NFS는 암호화가 되지 않는 서비스이다. 따라서 추가적인 NFS 보안강화를 위해, 물리적인 공인 IP 네트워크상에 사설 IP의 네트워크를 겹쳐서 구성한다. 즉 [그림 5]의 각 노드에 부여된 공인 IP의 네트워크는 그대로 사용하되, 각 노드에 IP 앨리어싱(aliasing)을 통하여 별도의 사설 IP를 부여한다. 그리고 서버측의 설정 파일 /etc/exports에 그 사설 IP 대역에 존재하는 NFS 클라이언트들만이 제공하는 파일 시스템을 마운트할 수 있도록 설정하면 그 설정을 모르는 한 불법적으로 마운트하여 사용하기 곤란하므로 보안이 향상된다.

4. 클러스터의 확장에 따른 성능 저하 및 교착 방지

4.1 목적 및 가정

[그림 5]처럼 확장된 클러스터에서 계산 전용 노드에는 과부하가 발생할 수 없으나 비전용 노드의 부하는 시시각각 변하며 과부하가 발생할 수 있다. 만일 이들 노드에 과부하가 발생하면 그 노드에서 실행중인 연산 프로세스는 인접 노드의 연산 프로세스와 즉각적인 메시지의 송수신이 불가능하므로 과부하가 해소될 때까지 다른 노드들도 대기하게 되어 클러스터의 성능이 저하된다. 심지어 어느 노드에 장애가 발생하면 연산 실패로 끝나게 된다.

본 논문은, 연산 도중 어느 프로세스의 과부하로 인해 전체가 교착 상태에 빠지거나 성능이 저하되는 것을 방지하기 위해, 지연된 프로세스는 일단 연산에서 제외하고 다른 프로세스가 그 역할을 나누어 연산을 수행하는 방안을 마련하고자 한다.

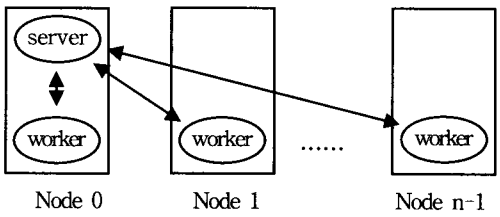
4.2 교착 방지를 위한 워커 주도의 전송방식

[그림 4]의 연산 구조는 이상에서 언급한 것과 같은 문제점을 가진다. 그 이유는

1) MPI 메시지 전송에서는, 송수신 당사자가 쌍을 이루어 각기 MPI_Send() 또는 MPI_Recv()와 같은 메시지 전달 함수를 동시에 호출한다. 따라서 한쪽의 반응이 없으면 상대방의 함수 호출은 완료되지 못하여 대기하게 된다.
2) Algorithm 1과 같은 동기 병렬 알고리즘은

메시지 송수신의 시점과 순서가 정해져 있으며 동기화되기 때문이다.

과부하 프로세스를 발견하고 조처를 취하기 위해서는, [그림 6]과 같이 워커(실제 연산을 수행하는 프로세스)의 상태에 영향을 받지 않고 워커 사이의 메시지 교환을 전담할 서버 프로세스를 별도로 둔다. 이는 별도의 노드에서 실행시킬 수도 있으나, 워커에 비해 작업량이 아주 적으므로 가장 성능이 강력한 전용 노드에서 워커와 함께 실행한다.



[그림 6] 메시지 패싱 구조

과부하가 걸린 워커 프로세스로부터 서버와 다른 워커 프로세스가 자유롭기 위해서는

1) 워커간의 데이터 교환은 반드시 서버의 중계를 통해서 하도록 한다.
2) 워커와 서버간의 메시지 전달은 반드시 워커의 주도하에 수행되어야 한다. 이렇게 함으로써 워커의 과부하에 의한 서버의 교착을 피할 수 있다.
3) 서버는 워커들로부터의 데이터 도착 순서대로 수신하도록 한다.
4) 일정 시간이 경과하여도 어느 워커의 요구가 없으면, 서버는 데이터 교환을 포기하고 그 워커는 일단 연산에서 제외하며 다른 워커들에게 작업을 재분배하여 연산을 계속한다.

MPI_Wtime()은 시스템 시간을 체크하는 MPI 함수이며, MPI_Iprobe()는 버퍼에 자신에게 전송된 메시지가 있는지 확인하는 MPI 함수인데, 인자로 MPI_ANY_SOURCE와 MPI_ANY_TAG을 사용하면 임의의 프로세스로부터 수신된 메시지가 있는지를 확인할 수 있다. [그림 7]은 이를 이용하여, 각 워커는 자신이 필요할 때 메시지(데이터 전송 혹은 요구)를 서버에게 보내고, 서버는 워커로부터의 메시지가 있음이 확인될 때만 수신 함수를 호출하여 수신된 메시지를 처리함으로써 프로세스들이 교착 상태에 빠지지 않도록 하는 핵심 알고리즘이다.

서버는 TOL*AvTime 시간동안 워커들로부터 무작위로 전송되어온 메시지를 확인하고 도착한 순서대로 처리한다. AvTime은 정상적인 상태에서 모든 워커가 Algorithm 1의 1회 루핑에 소요된 평균 시간으로, 측정된 소요 시간의 평균으로 택한다. TOL은 허용인자로 그 값은 임의로 적절히 정한다.

```
Stime = MPI_Wtime();
Received = 0;
do {
    MPI_Iprobe(MPI_ANY_SOURCE,MPI_ANY_TAG,
            communicator, &flag, &status);
    if (flag) { /* TRUE if a message has arrived. */
        송신자와 메시지 종류를 파악하고 메시지를 수신;
        Received++; }
    Elapsed = MPI_Wtime()-Stime;
} while ((Received < n) && (Elapsed < TOL * AvTime));
```

[그림 7] 교착 방지를 위한 서버의 수신 알고리즘

[그림 8]은 위 알고리즘에 의해 정해진 시간동안 모든 워커로부터 데이터를 수신하지 못한 경우, AvTime 시간 간격으로 최고 MaxTry 회수만큼 재시도한다. 그럼에도 불구하고 메시지를 보내지 않는 워커는 과부하가 걸린 것으로 판정한다.

```
for i = 1 to MaxTry, {
    MPI_Iprobe(MPI_ANY_SOURCE,MPI_ANY_TAG,
            communicator, &flag, &status);
    if (flag) { /* TRUE if a message has arrived. */
        메시지를 수신하고 송신자 및 메시지 종류 파악;
        break; }
    else
        sleep AvTtime;
}
```

[그림 8] 워커의 과부하 판정 알고리즘

일단 어느 워커의 과부하가 판정되면 그 워커에게 장애가 발생하였을 수도 있으므로, 막연히 그 과부하가 해소되기를 기다리는 것보다 해당 워커를 연산에서 제외하고 그 워커가 담당하던 작업을 다른 워커들에게 재분배하여 연산을 계속해 나가는 것이 효율적일 것이다.

4.3 2차원 격자 문제에서의 부하 재분배

성능 실험의 대상문제로서, 기상예보 모델 등에서 흔히 사용하는 2차원 격자문제의 예를 다루기로 한다. 어느 워커의 과부하가 판정되면 그 워커는 연산에서 제외하되 다른 워커로 하여금 과부하 워커의 격자선을 나누어 맡도록 한다.

예를 들어, 워커 1이 과부하로 판정되면, 워커 1은 연산에서 제외하고 워커 0과 워커 2는 각기 수직 격자선을 적절히(예를 들어 4개 및 5개)를 추가로 담당하도록 한다([그림 3] 참고).

4.4 교착 방지 알고리즘

이상의 기법을 격자문제에 적용할 경우, 워커와 서버의 알고리즘은 다음과 같다.

Algorithm 2 (워커)
서버가 Stop 신호를 보낼 때까지 반복한다.
1) 담당한 격자점의 값을 업데이트한다.
2) 서버와 경계치를 교환한다. 만일 서버가 부하 재조정의 명령을 내리면 Go to 1).
3) 수렴 판단에 필요한 정보를 계산하여 서버에게 보낸다.
4) 서버로부터 연산의 계속 여부에 대한 지시를 받는다.

Algorithm 3 (서버)
충분히 수렴할 때까지 다음을 반복한다.
1. 워커들의 경계치 메시지(Algorithm 2의 2)에서 보내는 것)를 탐지하고 수신한다.
   1) 만일 어느 워커가 보내지 않는다면 [그림 8]의 방법으로 일정 시간 수신을 시도한다.
      ① 수신에 실패하면 나머지 워커에게 작업 재분배 지시를 내리고 각 워커가 필요로 하는 데이터를 제공한 후 Go to 1.
      ② 만일 이상없이 수신되면 continue.
2. 각 워커들이 필요로 하는 경계치를 보낸다.
3. 각 워커들로부터 수렴 판단에 필요한 정보를 수신하여 수렴여부를 판단하고
4. 연산의 계속 여부를 워커들에게 알린다.

위 알고리즘은 이해를 돕기 위해 구조를 간단히 서술하였으나 설명을 생략한 부분이 있다.

첫째, 모든 워커는 자신의 경계치만을 서버에게 보낼 뿐 내부 격자점의 값은 혼자서만 유지한다. 그러나 과부하 워커의 작업을 나누어 맡는 워커는 내부 격자점에서의 값도 필요로 하므로 서버는 모든 격자점의 값을 유지해야 한다. 따라서 모든 워커는 정기적으로 내부 격자점의 값도 서버에게 보내어 서버가 전체 격자점의 값을 체크포인팅하도록 한다. 그러나 서버는 단순히 그 값을 메인 메모리에 저장할 뿐, [9] 등과 같은 복잡한 체크포인팅 방법을 사용하지 않는다.

둘째, 연산에서 제외된 워커는 영구히 제외하지 않고, 나름대로 연산을 수행하며 서버에게 보고토록 한다. 서버는 그 워커의 보고 간격을 확인해 정상적인 수준으로 회복된 것으로 판단되면 다시 연산에 참여시킨다.

MPI에서 모든 프로세스는 랭크(rank)로 구분되므로, 어느 워커를 연산에서 제외하거나 다시 참가시키는 것은 랭크를 명시함으로써 쉽게 구현된다.

5. 성능 실험

두 개의 계산 전용 노드에 LAN상의 노드를 포함하여 확장 구성한 클러스터는 <표 1>과 같다. node0와 node1은 계산 전용 노드이나, 나머지는

〈표 1〉 구성된 대칭형 클러스터

	공인/사설 IP의 끝 번호	기 종	BogoMips	비 고
node0	233	Pentium 4 3GHz	5931.00	계산 전용
node1	231	Pentium M 1.6GHz	1185.79	계산 전용
node2	230	dual Pentium 3 Xeon 1GHz	각 1974.27, 1994.75	실습용 서버
node3	180	Pentium 2 400MHz	790.52	웹 서버

고유의 역할을 수행하는 노드들이다.

모든 노드는 Fedora Core 4와 LAM/MPI v.7.1.1 -3을 사용하였다. BogoMips는 리눅스에서 제공하는 대략적인 성능을 나타내는 수치이며, node0의 성능이 가장 강력하므로 서버 프로세스를 이 노드에서 실행하였다.

성능 실험에는 각 큐의 상태의 수가 100개씩인 2큐 오버플로 모델 [8](이는 100×100의 2차원 격자 문제와 동일)의 해를 구하는 문제를 사용하였다. 비교를 위해 실험에 사용된 모든 노드들은 외부로부터의 웹 접속이나 로그인 등을 차단하였으며, 특정 워커의 과부하는 Algorithm 2를 일정 회수 반복한 후 일정 시간동안 시간을 소모하는 루프를 돌게 함으로써 시뮬레이션하였다.

병렬 프로그램은 연산도중 부하 재분배를 하더라도 단지 연산 범위의 조정에 의해 쉽게 담당 격자선을 가감할 수 있도록 작성하였다.

워커들은 각 노드당 하나씩 실행하되, 노드 2는 듀얼 CPU 시스템이므로 두 개를 사용하였다. 각 워커당 할당된 격자선([그림 3] 참고)의 수는 BogoMips를 참고하여 다음과 같이 고정하였다.

〈표 2〉 워커별로 할당된 작업량

워커 번호	0	1	2	3	4
노드 번호	0	1	2	2	3
할당 격자선 수	38	14	20	20	8

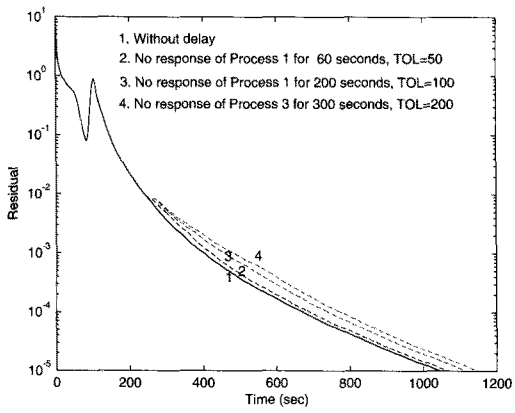
<표 3>과 [그림 9] 및 [그림 10]은 워커 1 또는 3이 일정 시간동안 반응이 없을 경우, 과부하로 판정하여 연산에서 제외하고 부하를 재분배하여 연산을 계속하는 경우의 소요시간 및 수렴 패턴을 보인다. 부하 재분배는 동적인 부하 재분배와 같은 복잡한 알고리즘을 사용치 않고 단순히 BogoMips를 참고하여 고정된 부하 배분 방식을 사용하였다.

<표 3>의 부하 재분배에서 1(14) → 0 (9), 2(5)는 1번 워커의 14개 블록을 0번과 2번 워커에게 각기 9개 및 5개 블록을 추가로 배정한 것을 의미한다. 그리고 연산에서 제외된 워커가 30회의 연산 동안 충분히 빨리 연산 결과를 보고할 경우는 다시 원래대로 부하를 재분배하여 연산에 참여토록 하였다. 부하 재분배시 필요한 내부 격자점에서의 값을 서버가 유지하기 위해, 모든 워커는 100회의 반복마다 자신의 모든 격자점의 값을 서버에

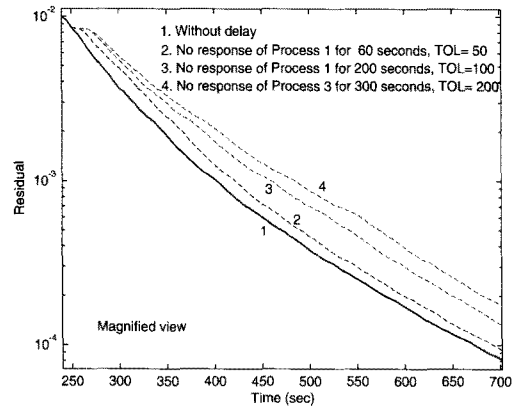
〈표 3〉 다양한 조건에 따른 소요 시간

결 과	과부하 워커	무반응 시간(초)	TOL	부하 재분배	소요 시간(초)
1	-	-	-	-	1,044.3
2	1	60	50	1(14) → 0(9), 2(5)	1,063.9
3	1	200	100	1(14) → 0(9), 2(5)	1,114.5
4	3	300	200	4(8) → 3(8)	1,147.1

게 보고하도록 하였다.



[그림 9] 다양한 조건에 따른 수렴 패턴



[그림 10] [그림 9]의 일부를 확대한 그림

[그림 9]는 다양한 조건에 따른 수렴 패턴을 그린 것이며, [그림 10]은 그 일부를 확대한 것이다. 수렴 패턴 1은 <표 2>처럼 부하를 분산하여 5개의 워커가 정상적으로 계산을 수행한 것으로, 계산 초기의 굴곡은 연산의 특성상 별 의미는 없다. 잔차의 크기가 10^{-5}에 도달할 때까지 대략 8,240회의 반복에 1,044초가 소요되었다. 수렴 패턴 2, 3, 4는 2,000회의 반복(약 251초 소요) 후 TOL의 값에 따라 정해진 시간동안 반응이 없는 워커를 연산에서 제외하고 그것의 부하를 재분배하여 연산을 하다가(적은 수의 워커가 작업하므로 수렴속도가 느려져 경사가 덜 가파르다). 그 워커가 정상으로 돌아와 다시 연산에 참여하는 경우(경사가 약간 더 가파른 부분)의 수렴 패턴을 나타낸다([그림 10] 참고).

만일 과부하 프로세스를 제외하지 않고 과부하가 해소할 때까지 기다렸다면 수렴 패턴 2, 3, 4의 경우 결과 1에 대기시간을 더한 만큼의 시간이 소요되었을 것이다. 그러나 본 알고리즘은 문제가 발생한 워커를 연산에서 제외하고 나머지 워커들만으로 비록 느리나마 연산을 계속하여 전체 시간을 상당히 단축하였음을 볼 수 있다. 그런데 1회 반복당 연산 시간이 너무 짧아(대략 1초에 8회 반복) 비교적 큰 TOL 값을 사용하였는데, TOL 값을 줄이면 빨리 연산을 재시작할 수도 있겠으나 약간의 지연에도 부하 재분배를 하게 되므로 오히려 성능이 떨어질 수도 있다.

6. 결론 및 향후 연구방향

본 논문에서는 전용 HPC 클러스터를 LAN상의 독자적인 서버를 포함하여 확장함으로써 컴퓨터 자원의 활용도를 최대한 높이는 방안을 제안하고, 그에 따른 문제점을 토의하고 해결책을 제시하였다. 즉 워커간의 메시지 전달 및 컨트롤을 담당할 서버 프로세스를 별도로 두어, 동기 병렬 연산 구조에서 교착 상태를 일으키는 원인을 제거하고 과부하 워커는 제외하되 그 부하를 재분배하여 연산을 계속하도록 하였다.

동적 부하분산이 목적이 아니므로 최적의 부하분산 전략을 고려치 않고 대략적인 CPU 성능에 따른 고정적인 부하 분산 정책을 사용하였으나, 콜드스타트(cold start)하는 보통의 체크포인팅/재시작과는 달리 중단없이 자동으로 수행한다는 점이 장점이라 하겠다. 또한 제안한 방법을 보다 명확히 설명하기 위해 격자 문제를 예제로 사용하였으나, 작은 문제로 분할이 가능한 모든 경우에 적용이 가능할 것이다. 사실 격자문제는 워커들의 모든 작업이 강결합(tightly coupled)되어 비동기 연산이 불가능하므로 프로세스 팜(process farm)

모델이 적용 가능한 문제들보다 병렬연산이 어렵다 하겠다.

한편 제안 방법은 넓은 의미로 [5]의 6가지 서버 가상화 형태중 분산부하관리(distributed workload management)에 해당한다 할 수 있으며, MPI로는 더 이상의 가상화가 불가능하다. 그러나 VMware Infrastructure[18] 같은 하이퍼바이저(hypervisor) 및 [4]와 같은 클러스터 파일 시스템을 활용하면 더 나은 결과를 얻을 수 있을 것이다.

참 고 문 헌

[1] 김영균, 오길호, "LAN 환경에서 유휴시간 예약에 기반한 PC Cluster 설계", 「한국정보과학회 2003년도 가을 학술발표논문집」, 제30권, 제2호(III), 2003.

[2] 김진미 외, "차세대 컴퓨팅을 위한 가상화 기술", 「ETRI 전자통신동향분석」, 제23권, 제4호(2008).

[3] 남기찬, 김용진, "서비스사이언스 관점에서 본 IT 서비스산업의 발전과제", 「소프트웨어진흥원 SW Insight 정책리포트」, 2008.

[4] 이규웅, "SAN 기반 클러스터 파일 시스템 SANique^{TM}의 성능평가 및 분석", 「한국IT서비스학회지」, 제7권, 제1호(2008), pp.195-203.

[5] Bittman, T., "The Future of Server Virtualization", Gartner Research Note T-20-4339, 2003.

[6] Fagg, G. E., E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, A. Bukovsky, and J. J. Dongarra, "Fault tolerant communication library and applications for high performance computing", *Proceedings of the Los Alamos Computer Science Institute Symposium 2003*, Santa Fe, NM.

[7] Gartner, Inc., "Gartner Identifies the Top 10 Strategic Technologies for 2009", http://www.gartner.com/it/page.jsp?id = 777212.

[8] Kaufman, L., "Matrix methods for queuing problems", *SIAM J. Sci. Stat. Comput.*, Vol. 4(1983), pp.525-552.

[9] Sankaran, S., J. M. Squyres, B. Barrett, and A. Lumsdaine, "The LAM/MPI Checkpoint /Restart Framework : System-Initiated Checkpointing", *International J. of High Perfornamce Computing Applications*, Vol.19 (2005), pp.479-493.

[10] Sloan, J., *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix, and MPI*, O'Reilly Media Inc. 2005.

[11] Subramaniyan, R., V. Aggarwal, A. Jacobs, and A. George, "FEMPI : A Lightweight Fault-tolerant MPI for Embedded Cluster Systems", *Proc. of International Conference on Embedded Systems and Applications (ESA)*, Las Vegas, NV, June 26-29, 2006.

[12] Zaki, M., W. Li, and S. Parthasarathy, "Customized Dynamic Load Balancing in a Heterogeneous Network of Workstations", In *5th IEEE Int. Symposium on High Perfornamce Distributed Computing*, 1996.

[13] Berkeley Lab Checkpoint/Restart(BLCR), http://ftg.lbl.gov/CheckpointRestart/CheckpointRestart.shtml.

[14] Berkeley NOW Project, http://now.cs.berkeley.edu/.

[15] Open MPI, http://www.open-mpi.org.

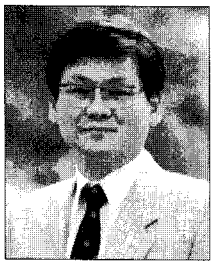
[16] PCs by day, supercomputer by night, http://www.theglobeandmail.com/servlet/story/RTGAM.20060615.tqsuperjun15/BNStory/GlobeTQ.

[17] http://www.linuxquestions.org/questions/linux-security-4/firewall-blocking-nfs-even-though-ports-are-open-294069/.

[18] http://www.vmware.com/kr/overview/.

[19] http://kr.blog.yahoo.com/thisrule1/857793.html.

◈ 저 자 소 개 ◈



박 필 성 (pspark@suwon.ac.kr)

서울대학교에서 이학사, 미국 Old Dominion University에서 응용수학 석사, 미국 University of Maryland at College Park에서 응용수학(응용분야 전산학) 박사를 취득하고 수원대학교 IT대학 컴퓨터학과 부교수로 재직중이다. 이전에 한국해양연구원에서 전산실장, 데이터센터장 및 수치모델연구그룹장을 역임한 바 있다. 관심분야는 리눅스 클러스터를 사용한 고성능 컴퓨팅, 병렬 처리, 수치해석, 리눅스 시스템 및 정보보안 등이다.