MPI 미들웨어에 기반한 병렬검색 시스템 구현

이정훈° 강미경
제주대학교 전산통계학과
{jhlee, mkkang}@cheju.ac.kr

An implementation of parallel search system based on MPI Middleware

Junghoon Lee Mikyung Kang
Dept. Computer Science and Statistics, Cheju National University

요 약

본 논문은 MPI 미들웨어에 기반하여 데이터베이스에 포함되어 웹에 의해 제공되는 정보들을 고속으로 검색할 수 있는 분산 병렬 검색 시스템을 구현한다. 지리적으로 산재한 막대한 양의 정보를 다루어야 하는 생물정보 분야 응용의 요구에 부합하기 위하여, LINUX를 탑재한 3 대의 PC로 구성된 클러스터를 구축하고 CGI 구동 프로그램, 마스터와 슬레이브로 구성된 MPI 프로세스를 구현하였으며 메시지 큐, MPI 프리미티브, HTTP 1.1 프로토콜에 의해 서로 통신한다. 마스터는 CGI의 요청에 따라 슬레이브에게 명령을 내려 동시에 해당 웹 페이지에 대한 검색을 수행하며 이를 통합하여 CGI에게 전달한다. 마스터는 다수의 CGI 요청들을 직렬화할 뿐 아니라 슬레이브들과의 동기화에 의해 최종적인 검색 결과를 수합한다. 본 논문에서 구현된 클러스터는 특정 어댑터의 추가 구현에 의해 새로운 데이터베이스에 대한 검색 기능을 추가할 수 있으며 동일한 운영체제와 미들웨어를 갖는 노드를 추가함으로써 협력 검색에 있어서 보다 많은 컴퓨터들을 참여시킬 수 있다.

1. 서 론

한 개의 CPU가 가질 수 있는 계산능력에는 한계가 있으며 이 한계를 극복하기 위해서 다양한 구조가 연구되어 출현하고 있다. 한 예로 멀티프로세서는 하나의 컴퓨터에 여러 개의 프로세서를 탑재하고 고속의 시스템 버스를 통해 연결하는데 이 방법은 많은 비용을 요구할 뿐 아니라 응용 프로그램의 개발에도 많은 어려움이 따른다[1]. 또다른 예로서 네트워크 기술의 발달은 여러 대의 컴퓨터를 네트워크로 연결하여 협력 처리하는 클러스터 컴퓨팅(cluster computing) 기법을 출현하게 하였다. 특히 영상 복원, 바이오 컴퓨팅 등 많은 수행시간을 필요로 하는 응용들의 출현이 예상됨에 따라 클러스터 컴퓨팅에 기반한 응용 개발과 재산기법의 연구가 많은 분야에서 진행되고 있다[2].

클러스터 컴퓨팅에 있어서 미들웨어는 서로 다른 노드에서 수행되는 병렬 프로세스들간에 효율적으로 자료를 공유하고 동기화되도록 하는 것으로서, 협력작업에 필요한 다양한 통신 프리미티브를 지원함으로써 시스템의 성능을 최적화할 수 있는 특징을 갖는다[3]. 제어구조와 데이터의 독립성이 강한 응용들의 요구사항도 반영할 수 있어야 하는데 현재 MPI(Massage Passing Interface)와 PVM(Parallel Virtual Machine)이 출현하였으며 이기종 컴퓨터간에 전송이 용이하고 전송시간이 빠른 MPI가 MPI가 국제표준으로 선정되었다.

현재 활발히 연구되고 있는 생명공학 분야는 막대한 양의 연구 및 개발 정보를 산출할 것으로 예상될 뿐 아니라 기존의 생물학에 관련된 정보 자체도 정보의 양이 막대한데 연구자 혹은 일반인 모두 이 정보들이 웹 상에서 제공되기를 원한다[4]. 이 정보들은 하나의 웹 서버에 저장되어 관리되기보다는 지리적으로 분산되어 저장될 가능성이 높다. 이와 같은 정보들 중에서 원하는 정보를 추출하는데 있어서 많은 검색시간이 필요하게 되므로 클러스터 컴퓨팅 기법을 도입한 검색 기법의 필요성이 대두되고 있다. 본 논문은 효율적인 분산 검색을 위하여 LINUX를 탑재한 PC들로 구성된 클러스터 상에서 웹을 통한 사용자의 검색 요구에 따라 각 MPI 프로세스들이 병렬적으로 검색하는 구조를 제시하고 구현한다. 이를 위해 웹 인터페이스, CGI 프로세스, MPI 프로세스들의 기능이 정의되고 이들간의 협력 구조가 제시된다. 본 논문의 구성은 다음과 같다. 1장에서 문제를 정의한 후 2장에서는 클러스터의 구성과 시스템의 구현 내용을 설명하고 3장에서는 각 구성요소간 상호작용에 대해 구체적으로 설명한다. 마지막으로 4장에서는 본 논문을 요약하고 향후 과제에 대해 기술한다.

2. 구 성

2.1 클러스터의 구축

표 1. 노드의 성능

HOST	CPU(Intel)	대역폭(Mbps)	메모리
노드1	Pentium-III 933	100(2)	256 M
노드2	Pentium-III 933	100	256 M
노드3	Pentium-II MMX 333	100	256 M

클러스터는 3대의 PC와 1대의 스위칭 허브로 구성되며 각 노드의 사양은 표 1에서 보는 바와 같고 전체 구조는 그림 1에서 보이고 있다. 노드1은 마스터 노드이며 랜 카드는 2 개

로 구성되어 있는 반면 나머지 2 개는 슬레이브 노드로서 각 노드는 LINUX 운영체제를 탑재하고 있으며 모두 MPI 서버가 수행된다. 마스터 노드가 관리 노드의 역할도 수행하게 되며 HTTP 프로토콜을 기반으로 웹 서비스를 제공한다[5]. 이 클러스터에는 새로운 노드가 추가될 수 있다.

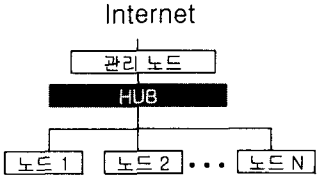

그림 1. 클러스터의 구성

2.2 프로세스의 구조

분산 웹 검색을 위한 프로세스들의 협력 구조는 그림 2에서 보는 바와 같이 클러스터상에 항상 수행중인 MPI 마스터와 슬레이브 프로세스, 사용자의 웹 요청에 따라 생성되는 CGI(Common Gateway Interface) 프로그램, 임의의 호스트에 위치할 수 있는 HTML 폼 스크립트 등으로 구성된다. 클러스터에는 MPI 프로세스가 항상 수행되고 있는데 사용자의 요청에 따라 drive.cgi가 활성화되면 질의어와 검색 결과가 전송된다. drive.cgi는 검색 결과를 HTML 형태로 사용자에게 도시하는 기능을 수행하고 요청 처리가 끝나면 자동으로 종료한다. 클러스터의 부팅과 더불어 마스터 및 2 개의 MPI 슬레이브들이(추후 확장 가능) 활성화되어 drive.cgi로부터의 요청을 처리한다. 이 과정에서 마스터와 슬레이브는 MPI 프리미티브를 이용하여 통신하며 해당 병렬 검색시 할당된 웹 페이지 URL에서 HTML 문서를 읽어와서 검색을 수행한다. 이때 각 요소간 협력은 기존의 운영체제나 웹서버에서 제공되는 프로토콜을 따라 수행되어야 한다.

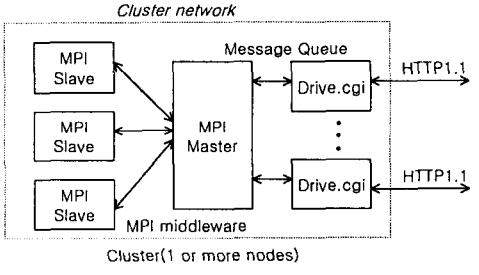

그림 2. 프로세스의 협력구조

3. 요소간 상호작용

3.1 사용자의 검색 명령 입력

폼 스크립트는 임의의 호스트에 위치하여도 무방하며 호스트의 운영체제에도 독립적이다. 본 논문의 구현에서는 chejutic.cheju.ac.kr/lic/lic5.php에서 스크립트를 제공하고 있으며, 인터페이스를 통한 사용자의 명령은 URL leto.cheju.ac.kr/jhlee/drive.cgi을 활성화시킨다. 동시에 다수의 사용자가 요청을 수행할 경우 HTTP 1.1 프로토콜에 의해 요청이 전달되어 새로운 프로세스가 leto에 생성된다. 동시에 생성될 수 있는 drive.cgi 의 개수는 시스템에서 정한 프로세스의 개수에 의해 결정된다.

3.2 MPI 마스터와 drive.cgi와의 연결

MPI 마스터는 하나의 프로세스로서 시스템의 초기화와 함께 수행을 시작되는 반면 drive.cgi는 사용자가 검색 메뉴를 선택할 때마다 하나씩 프로세스로 생성이 된다. 따라서 하나의 MPI 마스터에 여러 개의 drive.cgi 프로세스가 연결되어 있을 수도 있다. MPI 마스터와 drive.cgi들은 모두 하나의 컴퓨터 내에서 수행되는 프로세스들이므로 LINUX에서 제공하는 message queue를 사용하는 것이 효율적이다. UNIX 혹은 LINUX 계열의 운영체제에서 제공하는 message queue 메카니즘은 같은 호스트에서 수행되는 프로세스들 간에 운영체제 내의 버퍼를 이용하여 서로 통신할 수 있도록 하는 것으로서 응용 프로세스에서는 일련의 시스템 호출을 통해 통신 기능을 이용할 수 있다.

다양한 프로세스들 사이에 통신이 동시에 이루어질 수 있으므로 상대방을 정확하게 지정하고 원하는 메시지를 받을 수 있어야 하는데 이는 key와 message type에 의해 수행된다. 한 프로세스가 해당 key를 사용하여 운영체제 내에 메시지 큐를 생성하면 임의의 프로세스는 msgget 함수 호출에 의해 이에 대한 descriptor를 얻을 수 있으며 이 descriptor를 통해 msgrcv 혹은 msgsnd를 호출하여 운영체제 내 버퍼에 데이터를 쓰거나 읽는 기능이 지원된다. 이때 각 프로세스가 교환하는 데이터열은 처음 필드가 반드시 long(32비트 정수) 타입이어야 하며 이 타입 필드를 통해 원하는 메시지를 구분할 수 있다. 즉 msgrcv를 수행할 때 한 인자로서 수신하고자 하는 메시지 타입을 명세하면 해당 메시지만이 수신된다.

drive.cgi 들이 자신을 구분하기 위해 자신의 process id를 이용하며 이를 위해 MPI 마스터와 drive.cgi는 그림 3과 같은 자료구조를 기반으로 메시지를 교환한다. MPI 마스터와 drive.cgi들은 모두 위의 자료구조를 공유하고 있으며 MPI 마스터는 일단 0x5555 키를 이용하여 메시지 큐를 생성한 다음 mtype=1인 메시지가 도착하기를 기다린다.

```
struct t_buf {
    long mtype;
    long from;
    unsigned char buffer[MAXLEN];
}
```

그림 3. MPI 프로세스와 CGI 공용 자료구조

drive.cgi는 마스터에게 어떤 문자열에 대해 검색을 요청할 경우 mtype =1, from = 자신의 프로세스 id, buffer에 사용자가 입력한 문자열을 넣어 MPI 마스터에게 전송한다. mtype=1로 하면 MPI 마스터의 수신을 지정한다. 이때 drive.cgi는 msgsnd 호출을 하는 반면 MPI 마스터는 msgrcv 시스템 콜을 호출한다. 이때 drive.cgi는 자신의 프로세스 id를 알기 위해 getpid() 시스템 호출을 한다. MPI 마스터는 슬레이브들과 협력하여 문자열에 대한 검색을 완료

한 다음 msgsnd를 수행하고 drive.cgi는 msgrcv 함수 호출을 하여 검색 결과를 수신한다. drive.cgi에서 검색 결과를 수신하여야 하는 이유는 이 프로그램이 HTTP 1.1에 의해 시작된 프로그램이므로 이의 표준 출력(standard output)에 출력되는 내용이 사용자의 웹 브라우저에 보여지기 때문이다. 이 과정에서 MPI 마스터는 이전에 수신된 메시지의 from 필드를 저장하였다가 msgsnd 함수를 이용하여 drive.cgi에게 결과를 전달할 때 mtype 필드에 그 값을 넣는다. drive.cgi는 msgrcv를 할 때 기다리는 메시지 타입을 자신의 프로세스 id로 하고 있으면 다른 drive.cgi 들과 동시에 수행된다 하여도 전혀 메시지 송수신에 혼란이 생기지 않는다. 이때 MPI 마스터는 drive.cgi에게 4번의 msgsnd를 수행하는데 3 개의 검색 프로세스(마스터와 슬레이브)들이 검색한 결과 전송과 아울러 마지막으로 전송 결과의 종료를 알리는 메시지를 전송하는 과정으로 구성된다. 물론 슬레이브의 개수를 증가시키는 경우에도 이와 같은 프로토콜에 의해 효율적으로 분산 검색 수행이 가능하다.

3.3 MPI 마스터와 MPI 슬레이브 간의 연결

MPI 마스터와 MPI 슬레이브 간에는 MPI 미들웨어에서 제공하는 통신 프리티미브를 이용한 통신이 이루어진다. drive.cgi로부터 검색 요청이 수신되면 MPI 마스터는 MPI 미들웨어에서 제공하는 함수인 MPI_send()를 이용하여 슬레이브들에게 검색할 문자열을 전달한다. 반면 슬레이브들은 MPI_receive()를 이용하여 검색할 문자열을 수신하게 된다. MPI 그룹에 속한 태스크들이 명시적으로 통신을 하여야 하는 이유는 MPI가 공유 메모리를 지원하는 것이 아니고 메시지를 효율적으로 교환하는 매카니즘만을 지원하기 때문이다. 마스터와 슬레이브는 모두 하나의 프로그램(url.c)에서 생성되며 임의의 크기를 갖는 문자열을 교환할 수 있다. 물론 문자열을 구조체로 변경하면 일련의 자료구조들을 교환할 수도 있다. MPI 마스터는 외부 프로세스나 사용자들에게 보여지지 않으며 오직 사용자가 HTML 문서에서 구동시킨 drive.cgi의 메시지 전달에 의해서만 동작이 시작된다.

3.4 MPI 프로그램과 검색대상 웹 서버

MPI 프로그램은 기능적으로 MPI 부분과 HTTP 프로토콜 처리에 의한 해당 URL 접근 및 분석 기능으로 구성된다. URL 접근은 상대방의 웹 서비에게 URL을 요청하는 방식을 따르며 이는 일반적인 HTTP 클라이언트의 기능에 해당한다. 상대방 웹 서버와 연결하기 위해서는 트랜스포트 계층 위에서 동작하는 소켓 인터페이스를 사용하여 80번 포트에 connect을 수행한다. 이후 HTTP의 GET 명령 문법에 따사 write하면 웹 서비는 해당 페이지를 전송하게 되고 이는 read에 의해 수신이 가능하다.

수신된 웹 페이지에 대해 스트링 검색 혹은 링크 분석 등을 수행하여야 하는데 이는 각 페이지의 특성에 따라 달라지게 된다. 게시판, 장비목록, 생물정보 목록 등 본 논문에서 대상으로 하고 있는 대부분의 정보들은 데이터베이스에 저장되어 각 웹 서버가 구동하는 검색 프로그램에 의해 사용자들에게 전달되는 것이 일반적이다. 예를들면 chejutic.cheju.ac.kr/equip/srcitem.php?sname=##과 같은 URL에 의해 웹 서비에게 검색이 요청된다. 따라서 웹 서비와 연결이 된 이후에는 'GET /equip/srcitem.php?sname=##'를 write하고 read를 수행하면 서버로부터 그 결과를 전송받게 된다. 이를 위해서는 MPI 프로세스들이 자신에게 할당된 웹 페이지마다 URL과 함수이름 그 인자들에 대해 알고 있어야 한다. 또 수신된 페이지도 각 사이트마다 자신의 표현방식에 따라 다양한 정보들을 보이게 된다. 이 페이지 분석도 각 페이지에 맞추어 수행하도록 프로그램된다. 결국 검색 기능은 각 페이지의 특성을 많이 고려한 프로그램으로 작성되어야 하는데 이를 효율화하기 위해서는 각 페이지에 대한 어댑터 프로그램이 구현되어 MPI 프로세스들과 같이 컴파일된다.

어댑터 프로그램에서 read에 의해 웹 서버로부터 전달되는 페이지는 일단 파일로 저장된다. 사실상 약간의 페이지 내용을 보이기 위해 많은 문자열을 포함하는 HTML 문서가 웹 서버로부터 전달되므로 어떤 메모리 내의 변수에 웹 페이지의 모든 내용을 입시 저장하는 것은 불가능하다. 또 HTML 문서는 각 라인 단위로 처리하면 상당히 편리한데 read 함수는 라인을 구분하지 않고 네트워크를 통해 읽어 들이므로 분석 프로그램을 작성하는데 많은 어려움을 내포한다. 앞에서 언급했던 바와 같이 drive.cgi에서의 동시 검색 요청은 메시지 큐에 대한 대기 방식에서 직렬화되므로 마스터나 슬레이브들이 하나의 특정한 파일을 사용한다 하여도 동시접근에 의한 파일 접근 오류는 발생하지 않는다.

4. 결 론

본 논문에서 구현된 정보 통합검색 기능은 계속해서 확장이 가능하며 새로운 사이트기 검색 대상으로 선정될 경우 이 사이트에 대한 검색 그룹을 설정한 후 어댑터 프로그램 구현에 의해 웹 서비와 통신하고 페이지를 분석하도록 하여 규모의 확장을 가능하게 한다. 향후 본 시스템에 추가될 웹 사이트의 수는 국내 및 국외를 대상으로 계속 증가할 것이며 많은 양의 정보에 대해 보다 효율적인 검색을 수행할 수 있다. 본 논문에서 구현된 병렬 검색 프로토타입은 클러스터를 이루는 LINUX 계열 운영체제의 메시지 큐, MPI, 소켓 등의 기능을 기반으로 하고 있어서 확장성이 우수하다. 또 클러스터에 포함된 노드의 수를 증가시키고 MPI 슬레이브의 수를 증가시킨다면 많은 범위에 대한 동시 정보처리를 할 수 있도록 확장될 것이다.

참고문헌

[1] D. P. Bertsekas, J. N. Tsitsiklis, *Parallel and Distributed Computation*, Prentice Hall, 1989.

[2] I. Crawford, K. Wadleigh, *Software Optimization for High Performance Computing : Creating Faster Applications*, Prentice Hall, 2000.

[3] P. Pacheco, *Parallel Programming With MPI*, Morgan Kaufmann publishers, 1996.

[4] Thomas Down, *The BioJava Tutorial*, available at http://biojava.org/tutorials (The Open Bioinformatics Foundation), 2002.

[5] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *RFC 2616, Hypertext Transfer Protocol--HTTP/1.1*, Network Working Group, 1999.