그리드 환경에서의 데이터 관리를 위한 시스템

황상준° 노재춘
세종대학교 소프트웨어 대학원 freezer34@hanmail.net°
세종대학교 소프트웨어공학과 jano@sejong.ac.kr

A Data Management System For Grid Environment

Sangjun Hwang° Jaechun No
Dept. of computer software, Sejong University°
Dept. of computer software, Sejong University

요 약

과학 분야의 발전에 따라 처리해야 하는 데이터의 양도 급격하게 증가하여 기가바이트, 테라바이트 혹은 페타바이트 이상이 되었다. 이렇게 큰 단위의 데이터를 로컬영역에서 처리하기에는 무리가 있다. 본 논문에서는 테라바이트 혹은 페타 바이트 이상의 데이터를 처리하고 관리하는 방안과 데이터의 사용방법에 대해서 논의 하겠다.

1. 서 론

응용, 바이오, 나노, 항공우주 분야 등의 과학 분야의 발전에 따라 처리해야 하는 데이터의 양은 기가바이트, 테라 심지어 페타바이트 이상의 저장 공간을 필요로 하게 되었다.

테라 혹은 페타바이트 이상의 데이터를 저장하기 위해서 각 연구기관은 대용량 저장장치를 필요로 하게 된다. 하지만 대용량의 저장장치를 사용하기 위하여서는 고비용을 충당해야 하고, 로컬 영역에서 사용할 수 있는 용량에 대한 한계를 느끼게 된다. 이러한 문제를 해결하기 위한 방법으로 데이터를 분산하여 처리하는 방법이 있다.

데이터를 분산시켜 네트워크에 연결된 각 저장장소에 저장함으로써 저비용으로 대용량의 저장 장치를 구현할 수 있다. 본 논문에서는 대용량의 데이터를 저장과 관리하기 위한 방안과 함께 사용자의 요청이 있을 경우의 사용방안에 대해 논의 하겠다.

2장에서는 시스템을 구현하는데 필요한 기술에 대해 알아보고, 3장은 시스템의 특징, 4장은 시스템의 결과, 5장에서는 시스템에 대한 결론과 향후 발전과제에 대해 논의 하겠다.

2. 관련 연구

이번 장에서는 시스템을 구현하는데 필요한 기술들에 대해서 알아보기로 한다.

2.1 Message Passing Interface - Input/Output

MPI[1]의 표준은 60명의 사람과 40여개 기관의 협력으로 만들어졌다. 1992년 4월 29일부터 30일까지 버지니아에서 개최된 워크샵에서 메시지 전송에 대한 기본적인 개념의 정의와 표준을 위한 여러 가지가 정해졌다.

MPI는 공유된 메모리를 가진 머신 간의 통신에 이용되기도 하며, 서로 다른 개념과 구조를 가진 프로세서들 사이의 메시지의 통신도 가능하게 구성되어져있다.

또한 MPI-IO의 라이브러리의 일종인 ROMIO[2]는 여러 가지 파일시스템을 자동적으로 지원하여 사용자가 파일 시스템에 구애받지 않고 사용할 수 있게 지원해준다.

병렬처리 애플리케이션들은 입출력은 비연속적인 구조를 가지게 되고, 많은 수행을 원하게 된다. 이러한 액세스 패턴은 입출력에서 많은 오버헤드를 발생시키게 된다. 하지만 MPI-IO는 이러한 비연속적인 많은 수의 데이터들을 하나의 큰 액세스 패턴으로 제공함으로 인해 오버헤드를 줄여 성능을 향상시키는 역할을 하게 해준다.

2.2 Globus Toolkit[3]

1990년대 중반에 전문적인 과학과 엔지니어링을 위한 분산 컴퓨팅 구조인 그리드[4]가 시작되게 되었다. 그리드 컴퓨팅은 기존 상호간에 원하는 정보에 대한 서로의 합의 하에 이루어지는 공유의 개념이 아닌, 컴퓨터의 계산능력을 공유함으로써 보다 효율적으로 컴퓨터를 사용할 수 있게 하는데 있다. 이러한 그리드 컴퓨팅을 위해서 사용되는 툴로는 Globus Toolkit이 있다.

Globus Toolkit은 리눅스와 같은 오픈 소스 방식으로 개발되며 1998년 Globus 버전 1.0을 시작으로 2002년 2.0, 그리고 최근에는 Globus 버전 3.0 까지 개발되었다.

Globus Toolkit의 특징은 국가적, 지리적 제약을 벗어나 컴퓨팅 파워, 데이터베이스 외에 다른 툴을 사용하는 것으로, 자원의 모니터링, 자원의 관리 외에도 보안과 파일의 관리에 대한 서비스를 제공하는 것이다.

2.3 PostgreSQL[5]

우리가 PostgreSQL이라는 이름으로 알고 있는 목적 연관성 데이터베이스 시스템은 캘리포니아 대학에서 개발된 POSTGRES Version 4.2 패키지를 기본으로 개발 되었다.

PostgreSQL은 오픈 소스 데이터베이스이며 어디서든, 멀티 레벨을 동시에 제어할 수 있고, 대부분의 SQL문을 사용 할 수 있다. 또한, 여러 언어들(예를 들어, C, C++, JAVA, Perl, Python)을 지원한다.

3. GEDAS

GEDAS(Grid Environment-based Data Management System)는 데이터를 생성하거나 생성된 데이터를 원격지 시스템으로부터 로컬 영역으로 복사하기 위한 시스템으로 데이터를 생성할 때 사용되는 file layout과 메타데이터와 데이터 커뮤니케이션 부분으로 나눌 수 있다.

데이터를 생성할 때는 세 가지 방식의 file layout으로 생성하게 된다. 첫 번째는 independent file layout 이다. 시간마다 생성되는 데이터에 대해서 각각 다른 파일을 생성하여 저장하는 방식으로 파일 생성에 대한 오버헤드가 크다. 두 번째는 intermediate file layout 이다. 같은 그룹 안에서 같은 시간에 생성되는 데이터를 각 시간에 생성되는 파일에 저장함으로 파일 생성에 대한 오버헤드를 줄일 수 있다. 같은 파일에 저장한다 할지라도 메타데이터를 통해서 데이터를 구별할 수 있다. 마지막으로 long_length file layout 있다. 데이터를 생성할 때 같은 그룹 내에서 생성되는 데이터들을 시간에 관계없이 하나의 파일 안에 저장하는 방식으로 세 가지 방식 중에서 파일 생성에 대한 오버헤드가 가장 낮은 특징을 가지고 있다.

메타데이터는 데이터베이스에 연결하여 만약 테이블이 없는 경우에는 테이블을 먼저 생성한다. 그리고 데이터를 생성하게 되면 데이터에 대한 정보를 추가시켜 주는 역할을 하여 메타데이터를 생성한다. 데이터베이스는 원격지와 로컬영역에 각각 가지고 있다. 만약 어떤 데이터를 사용하고자 할 때에는 원격지와 로컬 영역에 있는 데이터베이스를 확인하여 데이터에 대한 메타데이터 값이 같다면 로컬 영역에 있는 데이터를 사용하게 되고, 로컬 영역의 메타데이터 값이 없거나 원격지의 메타데이터 값과 다르다면 데이터의 버전이 높은 쪽에서 낮은 쪽으로 데이터를 보내어 데이터의 버전을 일치 시켜 준다. 메타데이터를 관리하여 대용량의 데이터를 보다 효율적으로 관리할 수 있게 도와준다. 또한 데이터의 사용에 있어서 정보를 제공함으로 인해서 불필요하게 낭비되는 오버헤드를 줄일 수 있다.

데이터 커뮤니케이션 파트는 하나의 데이터를 사용하고자 할 때, 그 데이터를 계속 원격지에서 사용하게 된다면 환경 즉, 네트워크 트래픽과 데이터 오버헤드에 영향을 많이 받게 되는데, 이런 불필요한 요소를 제거함으로 인해서 데이터를 사용하기에 보다 유용하도록 로컬 혹은 좀 더 나은 환경으로 데이터를 복사하여 사용할 수 있게 해준다.

이제 메타 데이터와 데이터 커뮤니케이션 파트를 좀 더 상세히 살펴보자.

3.1 메타 데이터

메타 데이터의 정의는 데이터를 위한 데이터이다. 데이터에 대한 정보를 담음으로 데이터를 보다 쉽게 사용할 수 있게 할 수 있다. 특히 분산 환경에서의 메타데이터의 중요성은 로컬 환경에서의 메타데이터들 보다 더 큰 역할을 한다. 즉, 사용자가 사용하고자 하는 데이터를 위한 정보를 미리 가져옴으로 인해서 사용자가 원하는 데이터를 보다 쉽고 빠르게 찾아 사용할 수 있게 도와준다.

이 GEDDAS에서의 메타데이터는 데이터베이스로 구성되어 있으며, 분산되어 있는 데이터들의 정보를 저장하고 있다가 사용자의 요청이 있으면, 먼저 데이터베이스에 요청하여 메타데이터를 가져옴으로 인해서 데이터를 사용할 수 있게 도와준다.

또한 데이터를 새로 생성하거나, 기존의 데이터를 사용한 후 변경할 때 데이터베이스에 데이터의 정보를 자동으로 생성시켜주거나 변화시켜주게 된다. 이 때 로컬 영역에 있는 데이터베이스와 원격지에 구축되어 있는 데이터베이스는 항상 같은 정보를 담고 있어야 한다.

데이터베이스에서 사용되는 테이블은 5개로 구성되어 있으며 구성된 테이블은 다음과 같다.

① application registry table

어플리케이션에 대한 전반적인 메타데이터를 포함하고 있다. 데이터의 소유자가 아닐 경우, 처음 어플리케이션에 접근할 때 사용자에게 전달된다.

② data registry table

어플리케이션이 생선한 각 데이터에 대한 정보를 포함하고 있다.

③ file registry table

데이터 액세스를 위해 필요한 정보를 포함하고 있다.

④ system registry table

그리드상의 모든 시스템에 대한 정보가 포함되고, 해당 스토리지 상의 최적의 입출력 성능을 산출하기 위한 정보를 포함하고 있다.

⑤ performance registry table

각 리모트 영역에서 데이터 크기별로 시험한 데이터가 저장되어 있다.

3.2 데이터 커뮤니케이션

기본적으로 메타데이터의 정보를 바탕으로 데이터 커뮤니케이션이 이루어진다. 어떤 사용자가 특정한 데이터를 요구할 때, 메타데이터로부터 받아온 정보들을 기반으로 하여, 데이터가 어디에 있으며 어떤 형태의 파일로 되어 있으며, 버전이 어떻게 되며, 어떤 offset값을 가지고 있는지를 나타내어 사용자가 요청한 데이터를 효율적으로 사용할 수 있는 역할을 한다.

메타 데이터를 기본으로 하여 찾아지는 데이터는 기본적으로 MPI를 사용함으로써 병렬처리가 가능하게 되었다. <그림1>에서 보는 바와 같이 데이터를 분할하여 여러 프로세스를 동시에 사용함으로써 사용자에게 보낼 때 최대한의 속도로 보낼 수 있게 한다. 그렇다고 최대한의 프로세스를 사용하지는 않는데, 그 이유는 크기가 작은 데이터의 경우 분할하여 사용할 때 오버헤드가 커서 역효과를 가져 올 수도 있기 때문이다.

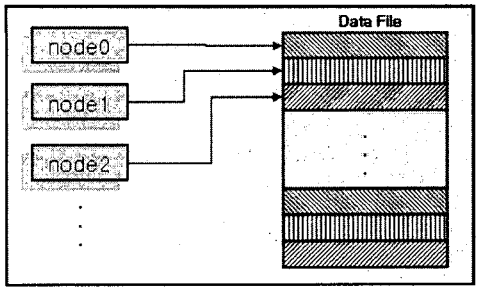

그림 1 데이터 파일의 처리

. 이와 같이, 분할되어 병렬 처리 되는 파일을 주고받을 때에는 소켓통신을 이용하게 된다. 소켓을 사용할 때에는 원격지와 로컬 간에 일대일 통신을 하게 되어 있다.

즉, 소켓을 사용하기 위해서는 포트를 열어야 하는데, 요청에 의한 데이터를 분리해야 하는 개수와 열리는 소켓의 포트의 개수가 같게 하여 전송함으로 인해서 효율성을 높일 수 있다.

4. 성능측정

성능 측정에 사용된 환경은 펜3-866CPU와 256M 메모리를 가진 컴퓨터에 RedHat Linux 9.0- 커널 버전 2.4.20-8 -을 설치한 4대의 컴퓨터를 100Mb 이더넷 허브를 사용하여 클러스터링 컴퓨터 환경을 구축하여 측정하였다.

성능 측정을 위해 시카고 대학에서 개발한 Astrophysics를 사용하였고, Block Data와 Cyclic Data 방식으로 1Gbyte를 생성하여 테스트하였다.

<그림 2>와 <그림 3>은 원격지에 있는 Block Data와 Cyclic Dada의 접근에 대한 성능을 나타낸다. 물론 그림에 나타난 그래프는 데이터에 접근하여 메타데이터를 얻어오는 시간이 포함되어있다.

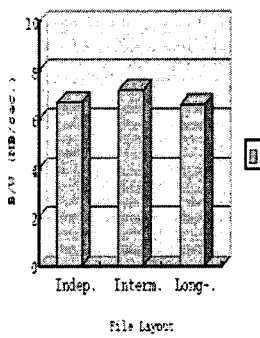

그림 2 원격지 Block Data

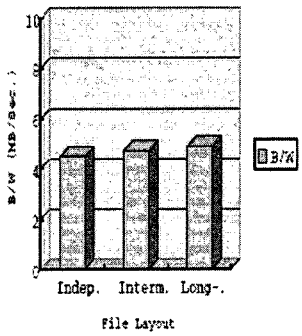

그림 3 원격지 Cyclic Data

<그림 4> 와 < 그림 5>는 복사하여 접근한 Block Data와 Cyclic Data에 대한 측정값을 나타낸다.

그림에서 사용한 Indep은 Independent file layout을 나타내며, Interm은 Intermediate file layout을, Long-은 Long -length file layout을 B/W는 BandWidth를 가리킨다.

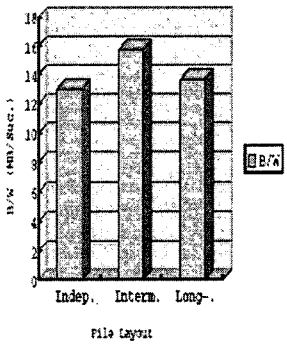

그림 4 복사된 Block Data

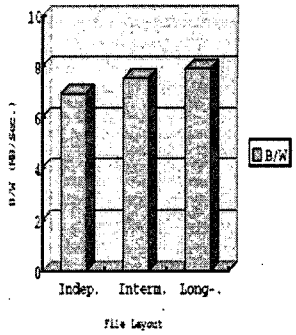

그림 5 복사된 Cyclic Data

5. 결론 및 향후 과제

성능 측정한 결과 복사된 데이터를 사용하는 것이 원격지에 있는 데이터를 사용하는 것 보다 더 좋은 성능을 나타내는 것을 볼 수 있다.

본 논문에서 제시한 GEDAS의 메타데이터의 경우 현재의 시스템은 데이터베이스를 로컬과 원격 사이트 영역에 두고 데이터 파일을 관리 할 수 있게 고안 되어 있다. 하지만 데이터베이스를 여러 곳에 구축하게 된다면 데이터베이스의 관리가 복잡해 질 수 있다. 이러한 문제점을 해결하기위해 데이터베이스를 중앙에 독자적으로 운영하여야 할 것이다.

데이터 커뮤니케이션은 소켓을 사용하기 때문에, 이더넷 환경에 종속적일 수밖에 없다. 소켓의 패킷 관리와 타이밍의 문제 등 데이터에 문제를 일으킬 수 있는 가능성이 있다. 패킷의 관리가 되지 않으면 완전한 데이터를 얻지 못하게 되어 데이터로서의 가치를 상실하게 된다. 이런 현상을 해결하고 보다 좋은 성능을 나타내기 위해서 데이터의 크기에 따라 커뮤니케이션에 사용되는 전송방식을 나누어 줄 필요가 있다. 즉 작은 데이터의 경우 소켓을 사용하고, 큰 데이터의 경우 RPC(Remote Procedure Call)이나 MPICH-G2[6] 혹은 GRIDFTP[7]를 사용하여 보다 빠르게 전송할 수 있다.

참고 문헌

[1] MPI, http://www.mpi-forum.org

[2] ROMIO, http://unix-unix.mcs.anl.gov/romio

[3] Globus, http://www.globus.org

[4] Ian Foster, "The Anatomy of the Grid", Intl j. Supercomputer Applikcations, 2001

[5] PostgreSQL Global Development Group, "PostgreSQL 7.3.2 Programmer's Guide", 1-22, 1996-2002

[6] MPICH-G2, http://www3.niu.edu/mpi/

[7] The University of Chicago and The University of Southern California, "Universal Data Transfer for the Grid", 2000.