CORBA 로드밸런싱 서비스를 사용한 범용 클러스터링 시스템

차현철° 최정호 김정선
한양대학교 컴퓨터공학과
{hccha°, choijh, jskim}@cse.hanyang.ac.kr

General-purpose Clustering System Using Standard CORBA Load Balancing Service

Hyunchel Cha° Jungho Choi Jungsun Kim
Dept. of Computer Science and Engineering, Hanyang University

요 약

인터넷과 개인컴퓨터가 대중화되어 사용자층이 넓어지고, 컴퓨터 기술이 다양화됨에 따라 점차 고가용성 서버에 대한 관심이 높아지면서 낮은 비용에 높은 성능을 만족시켜줄 수 있는 클러스터링 시스템이 인기를 끌고 있다. 하지만 기존의 클러스터링 시스템은 높은 성능에 비해 특정 플랫폼에 종속적이거나 특정 어플리케이션을 위한 구성이기 때문에 하나의 클러스터링 시스템을 여러 가지 플랫폼에서 다양한 목적으로 사용하기에는 부적절하다. 특정 장치 및 프로토콜 혹은 플랫폼에 의존적인 시스템은 유지보수 및 확장성의 제약을 받게 되기 때문에 이질적 컴포넌트들을 유기적으로 결합할 수 있는 방법이 요구되며 이러한 시스템을 효과적으로 구현하기 위해서는 표준 기반의 COTS (commercial-off-the-shelf) 미들웨어의 적용이 반드시 필요하다.

본 논문에서는 미들웨어로서 분산객체컴퓨팅의 표준인 CORBA 로드 밸런싱 서비스를 이용하여 유지보수 및 확장성이 용이하고 다양한 플랫폼에서 사용이 가능하며 여러 종류의 어플리케이션의 수행을 위한 클러스터 노드의 동적 추가/삭제가 가능한 범용 클러스터링 시스템을 설계하고 구현한다.

1. 서 론

지난 수년간 컴퓨터의 보급은 급속도로 이루어지고 있으며 학교나 기타 단체의 경우, 수십에서 수백 대의 컴퓨터들이 사용되고 있다. 이와 더불어 네트워크의 발달로 인해 속도는 빨라지고 비용은 계속 감소하고 있다. 이런 네트워크 기술의 성장으로 인해 그 이용분야가 넓어지고 사용자가 많아짐에 따라 네트워크 트래픽은 매년 평균 100% 성장률을 기록하고 있다[1]. 이에 따라 서버의 처리능력도 개선해야 하지만 실제적으로 처리능력의 개선에는 한계가 있다.

처리능력의 개선을 위한 한 가지 방법은 더 높은 성능을 가진 서버로 교체하는 것이나, 처리량이 증가할 때마다 서버를 계속 교체하는 하는 것은 비용이 너무 많이 든다는 단점이 있다. 다른 방법은 기존에 사용하던 다 수의 서버를 이용하여 클러스터로 묶어 많은 작업을 분산하여 처리하는 것이다. 이 방법은 낮은 비용으로 높은 성능 향상을 꾀할 수 있으므로 현실성이 높은 방법이라고 할 수 있다[2].

하지만, 다양한 서버와 다양한 플랫폼이 존재하고, 어떤 어플리케이션이 실행될지 모르는 상황에서 특정 플랫폼 혹은 특정 어플리케이션에 종속적인 클러스터링 시스템은 비효율적이라 할 수 있다.

그래서 본 논문에서는 CORBA를 이용하여 유지보수 및 확장성이 용이하고 다양한 플랫폼에서 사용이 가능하며 여러 종류의 어플리케이션의 수행을 위한 클러스터 노드의 동적 추가/삭제가 가능한 범용 클러스터링 시스템을 설계하고 구현하고자 한다. 2장에서는 클러스터링 시스템 특징과 미들웨어인 CORBA에 대해서 기술하고, 3장에서는 CORBA 기반의 범용 클러스터링 시스템을 설계 및 구현하며, 4장에서 테스트 결과를 마지막으로 5장에서 구현에 대한 결과를 정리하며 결론을 맺는다.

2. 기술 동향 및 관련 연구

2.1 클러스터링 시스템

클러스터링(Clustering)이란 여러 대의 서버를 연결하여 가변적인 업무부하를 처리하거나, 특정 서버에 문제가 발생했을 경우에도 운영이 계속되도록 하는 것을 말한다. 즉, 둘 이상의 컴퓨터를 마치 하나의 컴퓨터처럼 운용할 수 있도록 연결하는 것으로, 병렬처리 또는 부하, 배분, 고장 등의 사태에 효율적으로 대처하기 위한 방법이다. 일반적으로 클러스터링(Clustering)의 개념이 서버컴퓨터에 한정되는 경향이 있으나, 클러스터링(Clustering)은 PC나 워크스테이션 등 모든 기종의 컴퓨터를 사용할 수 있다. 클러스터링(Clustering)의 특징으로는 프로세서나 네트워크 장비, 그리고 저장장치 등으로 구성되고, 주로 오픈소스 기반의 운영체계와 소프트웨어를 사용하며, 가격 대 성능면에서 기존의 대형 컴퓨터를 능가하고, 업그레이드와 확장성이 우수하다는 것이다. 특징을 살펴보면

- 확장성 : 여러 서버를 클러스터링하여 서버 시스템을 구성하게 되어, 적은 비용으로 고성능 서버를 구축할 수 있어 많은 수의 클라이언트를 지원할 수 있다.
- 높은 가용성 : 클러스터링의 높은 가용성 덕분에 하나의 서버에 장애가 발생하여도 다른 서버를 이용하여 클라이언트들에 중단 없는 컴퓨터 시스템을 사용할 수 있다.
- 작업 부하의 균형적인 밸런싱 : 부하 조정을 이용하면 각 노드는 자신의 용량이나 로드에 맞게 요청을 처리할 수 있고, 클러스터 관리자에서 할당한 양의 프로세스를 처리할 수도 있다. 부하분산(로드밸런싱) 기능을 제공하기 때문에 서버 클러스터링 분산처리 또한 용이하게 해주어 클라이언트에게 빠른 서비스를 제공할 수 있다. 특히 서버들의 부하를 실시간으로 관찰해 상황에 따라 자동으로 클러스터 구성을 변경해주는 적응형 부하분산 기능을 제공하기 때문에 서버 자원의 활용도를 극대화함으로써 더 많은 클라이언트들에게 더 빠른 서비스를 제공할 수 있다.
- 관리기능: 서버 시스템 관리는 하나의 화면에서 클러스터링 환경 설정, 서버 및 네트워크 자원 모니터링 및 관리가 가능하기 때문에 전체 컴퓨터 시스템 관리도 용이하다.

2.2 CORBA의 구조

본 클러스터링 시스템은 미들웨어로서 분산객체컴퓨팅의 표준인 CORBA(Common Object Request Broker Architecture)를 사용하였다. CORBA는 개발언어나 네트워크, 운영체제 등 이기종 시스템간의 상호 연동을 가능하게 하는 미들웨어로서, 약 800여개 이상의 컴퓨터 관련 단체들이 참여한 OMG(Object

Management Group)에 의해 표준화 되었다[3].

CORBA는 컴퓨터 내부의 버스처럼 서로 다른 프로그램들 사이의 Bus 역할을 하는 모듈로서 각기 다른 언어로 구현되고, 또 분산되어 있는 모듈들에 대해 마치 로컬에 있는 것처럼 사용할 수 있도록 하는 기능을 제공해 준다.

최근 CORBA 3.0의 구조는 그림 1과 같다.

클라이언트에서 서버(구현 객체)로의 요청을 중개하는 것이 ORB(Object Request Broker)의 역할이며, ORB는 ORB core, Object Adapter, ORB Interface, Static IDL stub, Dynamic Invocation Interface(DII), Static IDL Skeleton, Interface Repository 및 Implementation Repository로 구성된다.[3]

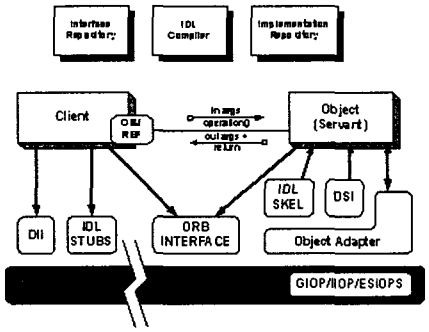

그림 1. CORBA 3.0의 구성

본 클러스터링 시스템은 CORBA 3.0을 만족하는 C++로 구현된 ACE 기반의 CORBA ORB인 TAO를 이용하여 구현하였다[4][5].

3. 범용 클러스터링 시스템 설계 및 구현

3.1 설계 고려 사항

범용의 클러스터링 시스템을 설계하고 구현하기 위해서 다음과 같은 사항들이 고려되어야 한다.

- 다양한 플랫폼에서 사용이 가능해야 한다.
- 클러스터 노드에서 다양한 어플리케이션의 수행이 가능해야 한다.
- 다양한 방법으로 작업의 분배가 가능해야 한다.
- 클러스터링 시스템에 클러스터 노드의 동적 추가/삭제가 가능해야 한다.

3.2 시스템 설계

다양한 플랫폼에서 사용이 가능하고 유지보수 및 확장성이 용이한 클러스터링 시스템을 구현하기 위하여 이질적인 컴포넌트들을 유기적으로 결합하고 다양한 플랫폼에서 사용할 수 있도록 만들어 주는 미들웨어로서 CORBA를 사용하고 C++로 구현된 ACE기반의 CORBA ORB인 TAO를 채택하였다.

또한 CORBA의 로드밸런싱 서비스(LoadBalancing Service)를 사용하고 추가로 정책을 구현함으로서 다양한 방법으로 작업들이 파일 형태로 분배가 될 수 있도록 설계하였고, 이렇게 제공된 파일형태의 작업을 분석하여 이를 수행할 수 있는 어플리케이션만 클러스터 노드에 구현해 주면 필요에 따라 다양한 어플리케이션용 클러스터링 시스템으로의 확장이 가능하다.

마지막으로 TAO의 Cygnus를 활용함으로써 클러스터 노드의 동적 추가 및 삭제가 가능하도록 설계하였다[7].

3.3 범용 클러스터링 시스템의 전체 구조

본 논문에서 구현한 범용 클러스터링 시스템의 구조는 그림 2와 같다. 특정 어플리케이션을 수행하기 원하는 사용자는 Web Server를 통해 어플리케이션에 필요한 데이터를 입력하고 이를 Facade Job Manager에서 시스템 내부에서 유일한 아이디를 생성하여 작업의 기본 단위인 하나의 Job으로 만들어 Execution Job Manager에게 제출한다.

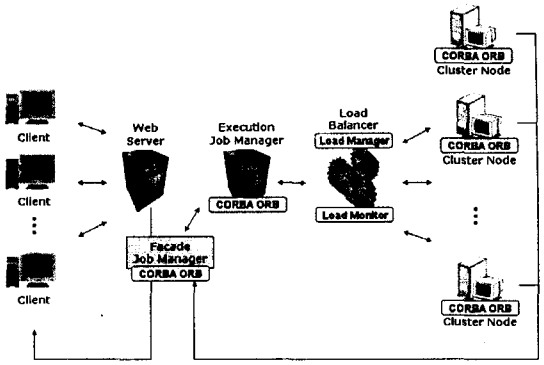

그림 2. 범용 클러스터링 시스템의 구조

이렇게 제출된 Job은 최초 구동시 클러스터링 시스템에 세팅되었던 방식으로 Load Balancer에 의해 특정 어플리케이션을 수행할 수 있는 Cluster Node로 분배 되고,수행이 완료된 Job은 CORBA의 Event Service를 사용하여 Facade Job Manager로 전송하게 되며 Facade Job Manger에서는 이를 다시 클라이언트에게 전달한다[6][7][8].

3.4 Facade Job Manager

Facade Job Manager의 구성 요소는 그림 3과 같다.

Job Generator는 클러스터링 시스템 내에서 유일하게 식별되는 Job 아이디를 생성하고 우선순위를 지정하여 클라이언트로부터 제출된 데이터를 통합하여 Excution Job Manager에게 제출한다. Completion Job Table Manager는 완료된 작업에 대한 데이터를 유지하며, 클라이언트 혹은 관리자의 요청시 제출된 작업의 상태를 보고한다.

Completion Job Handler는 특정 어플리케이션의 실행결과를 이벤트 채널을 통하여 수신하여 클라이언트가 요구하는 형식으로 어플리케이션의 수행결과를 전달한다.

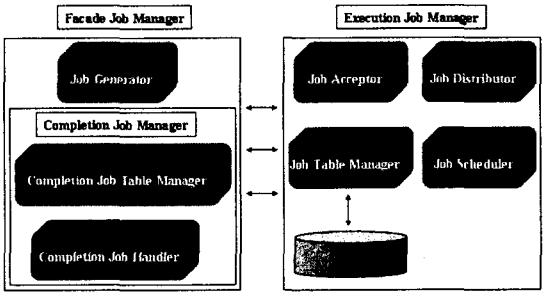

그림 3. Facade Job Manager 와 Execution Job Manager

3.5 Execution Job Mananger

Execution Job Manager의 구성요소는 그림 3과 같다

Job Acceptor는 Facade Job Manager로부터 Job을 접수하며, Job을 Job Table Manager를 통해 Job Table에 저장한다.

Job Table Manager는 Job Table을 관리하며, Job의 저장 및 삭제와 처리 상태를 관리한다.

Job Scheduler는 Job의 우선 순위(priority)에 따라 분배할 Job을 선택한다.

Job Distributor는 Job Scheduler가 선택한 Job을 로드 밸런서를 통해 클러스터 노드로 분배한다.

전체적으로 Job을 접수받아 분배되는 순서는 그림4와 같다.

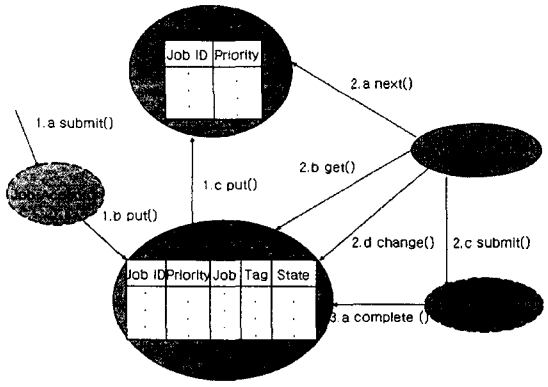

그림 4. Execution Job Manager에서의 Job 처리순서

3.6 Load Balancer

CORBA ORB와 Load Balancing Service는 TAO를 이용하여 구현하였다.

TAO에서 제공하는 로드 밸런싱 정책이외에 특정 어플리케이션에서는 하나의 Job이 완료된 후에 새로운 Job을 받아 처리하는 것이 더 효율적이므로 추가로 Customize 방식을 구현하였다. Customize 방식은 유휴상태의 클러스터 노드에게 새로운 Job을 분배하는 것으로 모든 클러스터 노드가 Busy 상태가 되게 되면 분배를 대기하다가 다시 유휴 상태의 클러스터 노드가 생기면 Job을 재분배 한다.

본 클러스터링 시스템에서 지원하는 로드 밸런싱 정책은 표 1과 같다.

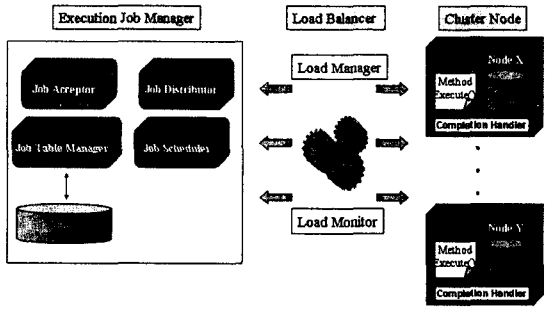

그림 5. Load Balancer와 Cluster Node

정 책	내 용	
Round Robin	차례대로 클러스터 노드에게 분배	
Random	임의의 클러스터 노드에게 분배	
Least Loaded	CPU	CPU 사용량이 가장 적은 클러스터 노드에게 분배
	Disk	Disk 사용량이 가장 적은 클러스터 노드에게 분배
	Memory	Memory 사용량이 가장 적은 클러스터 노드에게 분배
	Network	Network 부하가 가장 적은 클러스터 노드에게 분배
	Customize	유휴 상태의 클러스터 노드에게 분배, 기본으로 Random 방식으로 분배하다가 모든 클러스터 노드가 busy 상태가 되면 분배를 대기

표 1. 로드 밸런싱 정책

3.7 Cluster Node

클러스터링 시스템의 실행 중에 클러스터 노드가 등록되고 실행이 종료된 후에 삭제하도록 함으로써 동적인 추가 및 삭제가 가능하다.

본 클러스터 노드에서는 특정 어플리케이션을 실행할 수 있는 모듈을 만들고 제출받은 Job을 수행할 수 있도록 작성한다. Job이 완료되면 수행 결과를 Facade Job Manager의 Completion Job Handler에게 전송한다.

4. 실험 및 결과분석

4.1 실험환경

운영체제 : Windows2000, Linux, Solaris8

어플리케이션 : 레디언스 엔진(Radiance Engine)

4.2 실험결과

현재 개인용 PC와 서버에서 많이 사용되는 Windows2000과 Linux, PC용 SunOS 인 Solaris8을 이용하여 본 시스템을 테스트 해 보았다. 레디언스 엔진(Radiance Engine)은 미국 Lawrence Berkeley Laboratory(LBL)에서 개발된 프로그램으로 빛의 거동을 물리적으로 시뮬레이션한 결과로부터 조도 및 휘도 분포를 계산하고 가시화 하여주며 빛 환경의 정량적, 정성적 평가를 하는 데 사용이 되어진다[9]. 본 실험에서는 Linux와 Solaris8 운영체제하에 PC와 서버에서 클러스터 노드들의 응용 어플리케이션으로 레디언스 엔진을 수행하고 클러스터 노드의 동적 추가/삭제가 가능한지를 테스트하고, Windows 2000에서 Execution Job Manager를 실행함으로써, 다양한 플랫폼에서 다양한 정책으로 로드 밸런싱이 가능한지 테스트를 수행해 보고 가능함을 확인하였다.

5. 결론 및 향후 연구 과제

다양한 플랫폼과 어떤 종류의 어플리케이션이 수행될지 모르는 상황에서 특정 플랫폼 혹은 특정 어플리케이션에 종속적인 클러스터링 시스템은 유지보수 및 확장성에 제약을 받게 된다.

본 논문에서는 플랫폼과 특정 어플리케이션에 영향을 받지 않고 다양한 용도로서 클러스터링 시스템을 손쉽게 사용할 수 있도록 범용 클러스터링 시스템을 설계하고 구현하였다.

향후과제로서 기존 시스템에 클러스터링 시스템의 상태를 효율적으로 관리하기 위한 모니터링 시스템과 이상 발생시 자동으로 클러스터링 시스템을 복구할 수 있는 기능을 추가한다면 더 효율적인 클러스터링 시스템을 이룰 수 있을 것으로 기대된다.

참고문헌

[1]Wensong Zhang, Shiyao Jin, Quanyuan Wu "Creating Linux Virtual Servers", LinuxExpo 1999.
[2]David A. Patterson, John L. Hennessy "Computer Organization & Design", Mogan Kaufmann
[3]www.omg.org
[4]"http://www.cs.wustl.edu/~schmidt/"
[5]Douglas C. Schmit, David L. Levine, and Sumedh, "The Design and Performance of Real-time Object Request Brokers", Computer Communications, 21(4):294-324, April 1998
[6]"Proposed CORBA Load Balancing and Monitoring Specification", Object Manager Group, OMG Document mars/02-10-14 edition, October 2002
[7] Ossama Othman, Carlos O'Ryan, and Douglas C. Schmidt, "Designing an Adaptive CORBA Load Balancing Service Using TAO", IEEE Distributed Systems Online, 2(4), April 2001
[8]Ossama Othman, Douglas C. Schmidt, "The Design and Performance of a Scalable CORBA Load Balancing and Monitoring Service", 2002
[9]Ward, G. J, "The RADIANCE Lighting Simulation and Rendering System", Computer Graphics, Proceedings of '94 SIGGRAPH conference