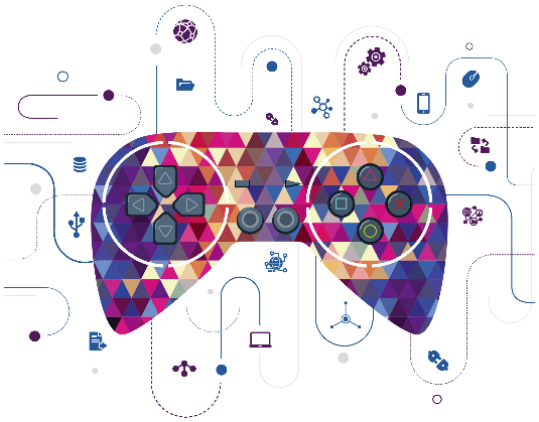


그림으로 배우는 구조와 원리

운영체제 개정 3판

Chatpter 11
분산 및 다중(병렬) 처리 시스템

- 분산 컴퓨팅의 목적과 특징
- 분산 및 네트워크 운영체제
- 다중 처리 시스템

Section 01 분산시스템(1. 네트워크와 분산 시스템)

네트워크와 분산 시스템의 개념

- 운영체제에서 분산 처리는 컴퓨터 사용자 간에 서로 데이터를 교환하여 처리할 수 있도록 네트워크로 상호 연결한 것
- 네트워크로 연결한 시스템은 사용자의 액세스를 제어하여 편리하게 자원 공유
- 네트워크로 연결한 시스템은 분산 시스템과 다중 처리시스템으로 구분
  - 분산 시스템 : 메모리와 클록을 공유하지 않고 지역 메모리를 유지하는 프로세서로 구성

    서로 독자적으로 동작
  - 다중 처리 시스템 : 하나 이상의 프로세스로 구성되며, 프로세스들이 메모리와 출력을 공유

    병렬 처리 시스템이라고도 함

2. 네트워크의 구성

네트워크의 개념

- 서로 독립된 시스템 몇 개가 적절한 영역 안에서 속도가 빠른 통신 채널을 이용하여 상호 통신할 수 있도록 지원하는 데이터 통신 시스템
- 컴퓨터 네트워크는 1960년대 사이트 간 효율적 통신 위해 학교 연구 프로젝트로 탄생
- 광범위한 사용자 모임 간에 하드웨어나 소프트웨어를 편리, 경제적 공유 지원
- 알파넷ARPANET :최초로 개발된 네트워크(1968년에 처음 작동)
- 사용자가 원거리의 하드웨어나 소프트웨어 자원에 액세스할 수 있는 기능 제공
- 네트워크 시스템 구성하는 방법
  - 강결합
  - 약결합

2. 네트워크의 구성

■ 강결합tightly coupled 시스템

- 프로세서들이 메모리를 공유하는 다중 처리 시스템
- 프로세서 간에 공유 메모리를 이용하여 통신하므로 공유 메모리를 차지하려는 프로세서 간의 경쟁 최소화해야 함
- 프로세서 간의 경쟁은 결합 교환combining switch 방법으로 해결
  - 하나의 공유 메모리 차지하려는 여러 프로세서 중 오직 하나의 프로세서만 액세스 허용하는 것

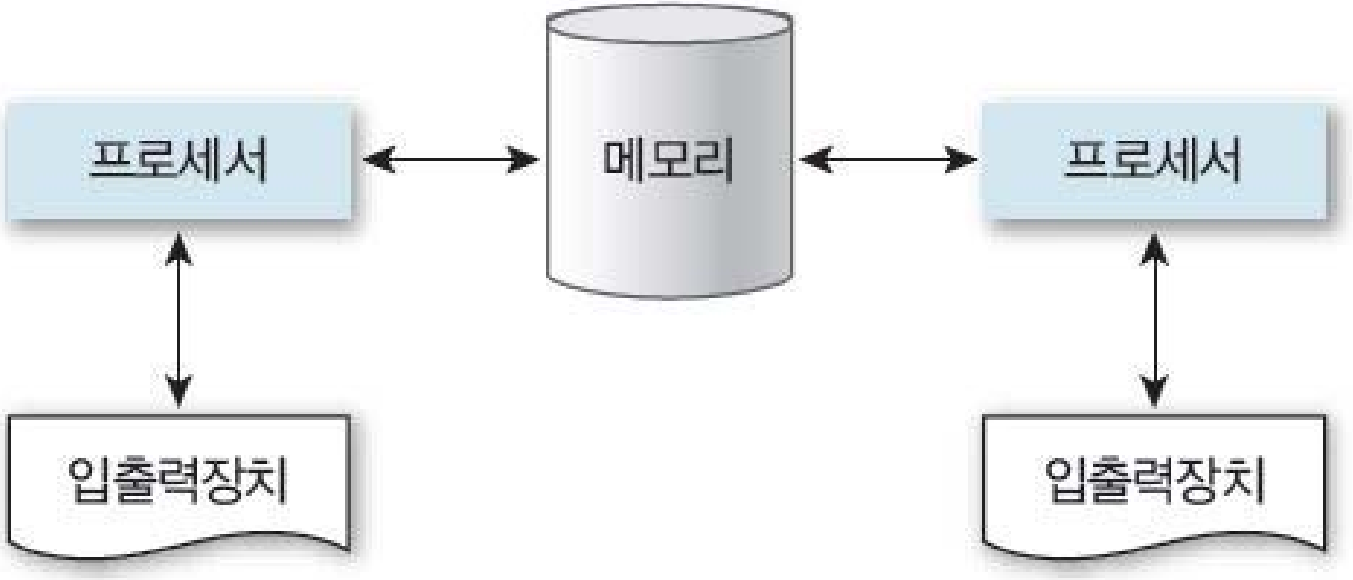


그림 11-1 강결합 시스템

2. 네트워크의 구성

■ 약결합loosely coupled 시스템

- 둘 이상의 독립된 시스템을 통신선으로 연결
- 각 시스템은 자신만의 운영체제, 메모리, 프로세서, 입출력장치 등 있어, 독립적으로 운영
- 필요할 때 통신선을 이용하여 메시지 전달이나 원격 프로시저 호출로 통신
- 통신선으로 다른 시스템의 파일 참조, 각 시스템의 부하 조절 위해 부하가 적은 프로세서에 작업 보낼 수도 있음. 하나의 시스템에서 장애가 발생해도 다른 시스템의 프로세서를 독립적으로 수행할 수 있어 치명적인 시스템 장애 발생하지 않음

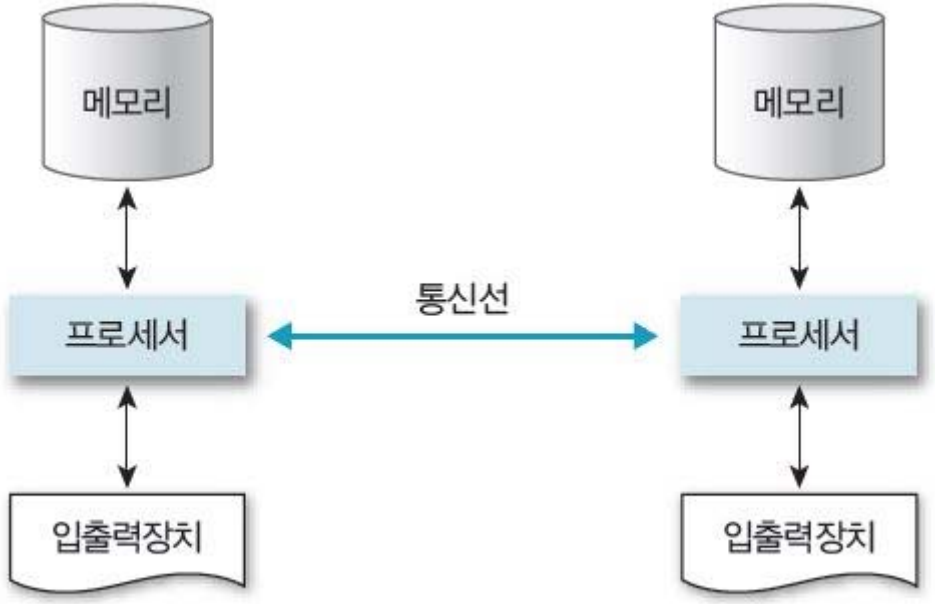


그림 11-2 약결합 시스템

3. 네트워크의 구조

망mesh 구조

- 각 노드를 시스템의 모든 노드와 직접 연결하는 완전 연결fully connected 방법
- 두 노드 간에 직접 통신선이 있어야 하므로 초기 설치비 많음
- 메시지 전달할 때 두 노드 연결하는 링크 하나만 매우 빠름
- 많은 링크가 고장이 나야 시스템 분할하므로 신뢰성 매우 높음
- 완전 연결 네트워크

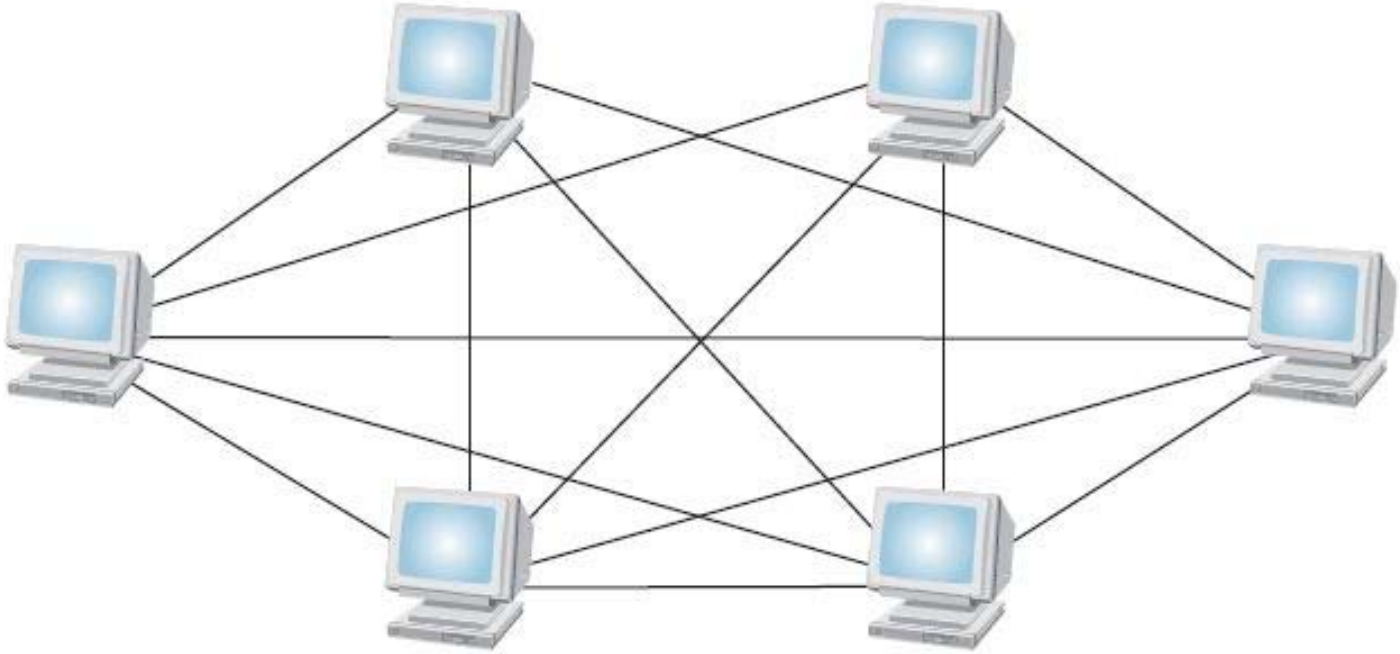


그림 11-3 완전 연결 네트워크

3. 네트워크의 구조

트리tree 구조(계층hierarchy 구조)

- 회사의 컴퓨터 네트워크에 사용하는 방법, 네트워크의 각 노드가 트리로 구성
- 루트 A를 제외한 각 노드는 단일 부모와 자식 몇 개를 가짐
- 기본 비용은 일반적으로 망 구조보다는 낮음
- 부모 고장이 나면 그 자식들은 서로 통신 불가, 다른 프로세스와도 통신 불가
- 트리 구조 네트워크

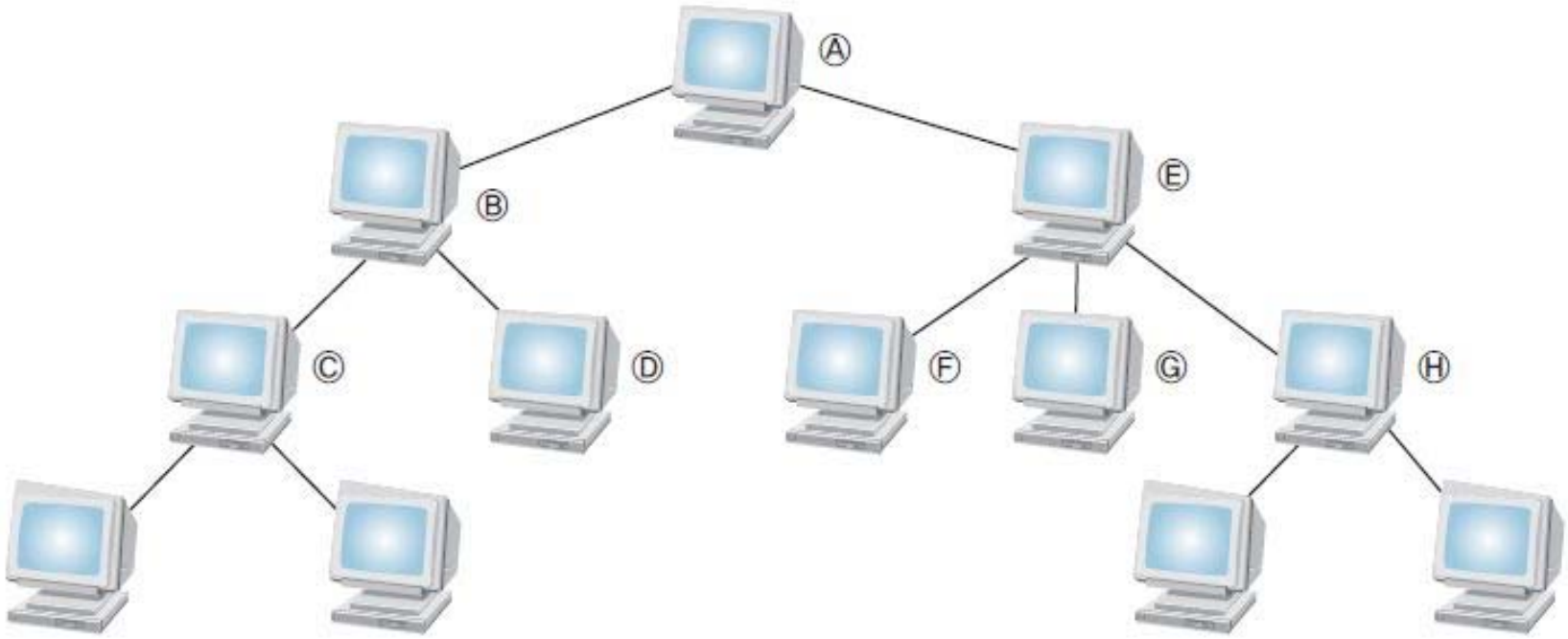


그림 11-4 트리 구조 네트워크

3. 네트워크의 구조

성형star 구조

- 모든 노드 중앙 노드에 직접 연결, 중앙 노드 외의 다른 노드 서로 연결하지 않음
- 중앙 노드가 메시지 교환 담당
- 중앙 노드에서 병목 현상이 발생하면 성능 현저히 떨어짐, 중앙 노드에 장애가 발생하면 전체 시스템 마비
- 기본 비용이 노드 수에 비례, 간단한 구조, 통신 비용 저렴, 집중 제어로 유지 보수 용이

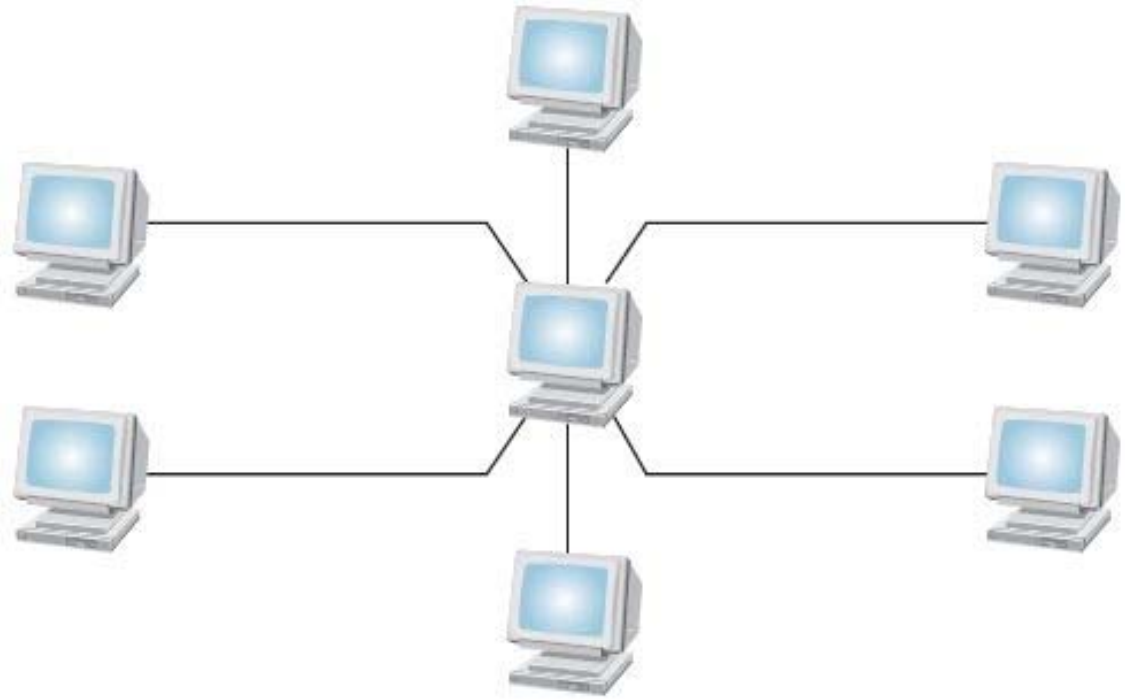


그림 11-5 성형 구조 네트워크

3. 네트워크의 구조

링ring 구조

- 각 노드를 정확히 다른 노드 2개와 연결하는 방법
- 메시지 전달 방향을 단방향 또는 양방향으로 구현
  - 단방향 구조는 한 노드의 한쪽 이웃 노드에만 전달, 양방향 구조는 한 노드가 양쪽 노드 모두에 전달
- 단방향 구조에서는 한 노드나 링크가 고장 나면 네트워크 분할
- 양방향 구조에서는 연결이 2개 고장이 나면 네트워크 분할
- 기본 비용이 노드 수에 비례, 메시지가 링을 순환할 때 통신 비용 증가

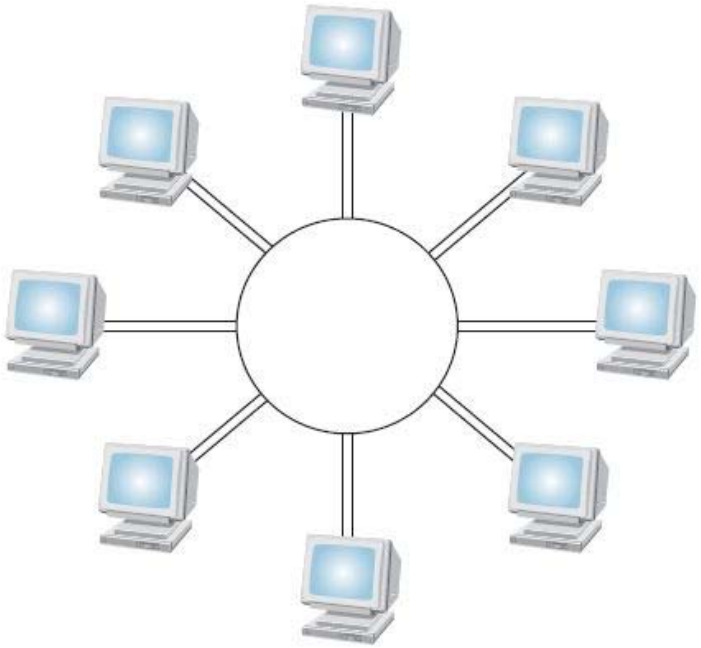


그림 11-6 링 구조 네트워크

3. 네트워크의 구조

버스bus 구조

- 연결 버스(중앙의 통신회선) 하나에 모든 노드 연결 방법
- 네트워크의 기본 비용은 노드 수에 비례, 버스를 공유하여 경제적
- 링크가 고장 나면 모든 노드 간의 통신 불가능
- 각 노드의 고장이 나머지 노드 간 통신에는 영향 주지 않아 신뢰성 높고, 노드의 추가·변경·제거 등이 비교적 쉬워 확장성 좋음
- 단점 : 버스에 장애가 발생하면 네트워크 전체에 영향을 미친다는 것

그림 11-7 버스 구조 네트워크

4. 원격 프로시저 호출RPC, Remote Procedure Call

원격 프로시저 호출의 개념

- 분산 시스템과 단일 시스템의 가장 큰 차이는 프로세스 간 통신이다. 단일 시스템에서 프로세스들은 공유 메모리로 입출력하여 서로 통신 가능하나, 분산 시스템은 공유 메모리 없음
- 분산 시스템에서 프로세스 간 통신은 클라이언트와 서버의 요청·응답 형태로 구현하는 대표적인 예
- Birrell과 Nelson이 제안
- 한 컴퓨터에서 실행하는 프로세스를 다른 컴퓨터에서 실행하는 프로세스의 프로시저(또는 함수)가 호출할 수 있게 하는 것으로, 클라이언트/서버 모델을 전제로 함
- 클라이언트와 서버 사이에 프로시저를 호출하는 프로그램을 허용하여 지역 프로시저를 호출해서 지역적으로 분산된 원격 프로시저 호출
- 클라이언트에 있는 프로세스가 서버에 있는 프로시저를 호출하면 클라이언트에 있는 프로세스는 중단하고 서버에 있는 호출된 프로시저 실행
- 함수의 반환 값이 네트워크 이용하여 클라이언트로 전송

4. 원격 프로시저 호출RPC, Remote Procedure Call

- 일반적인 RPC 단계(요청과 응답 관계 일대일(1:1))

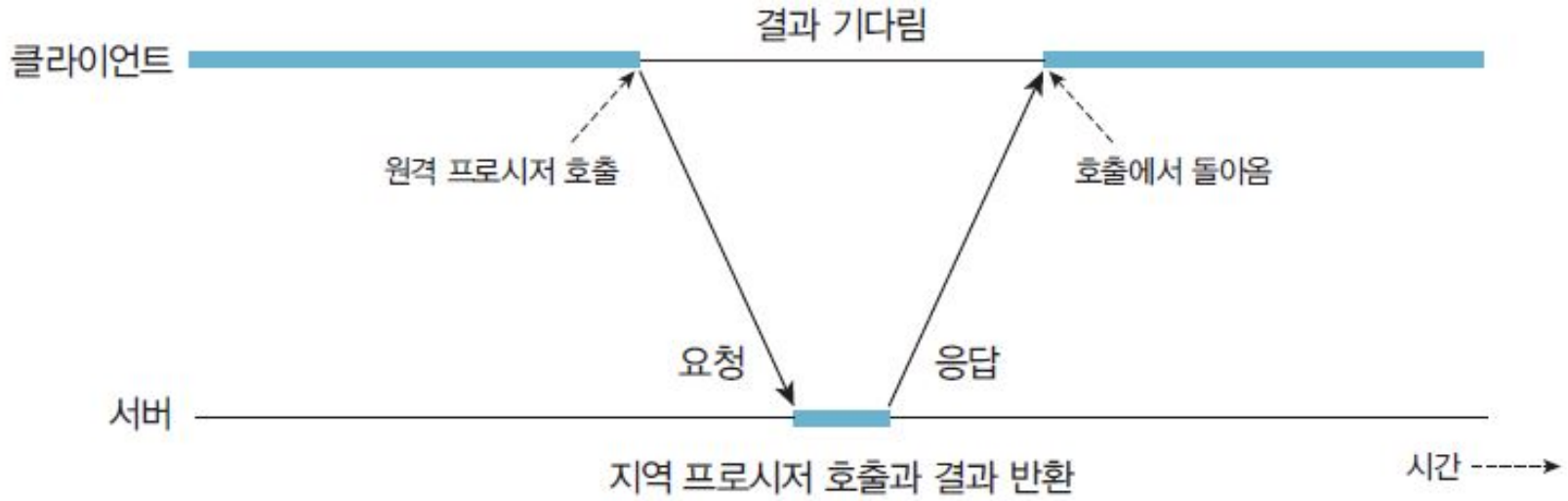
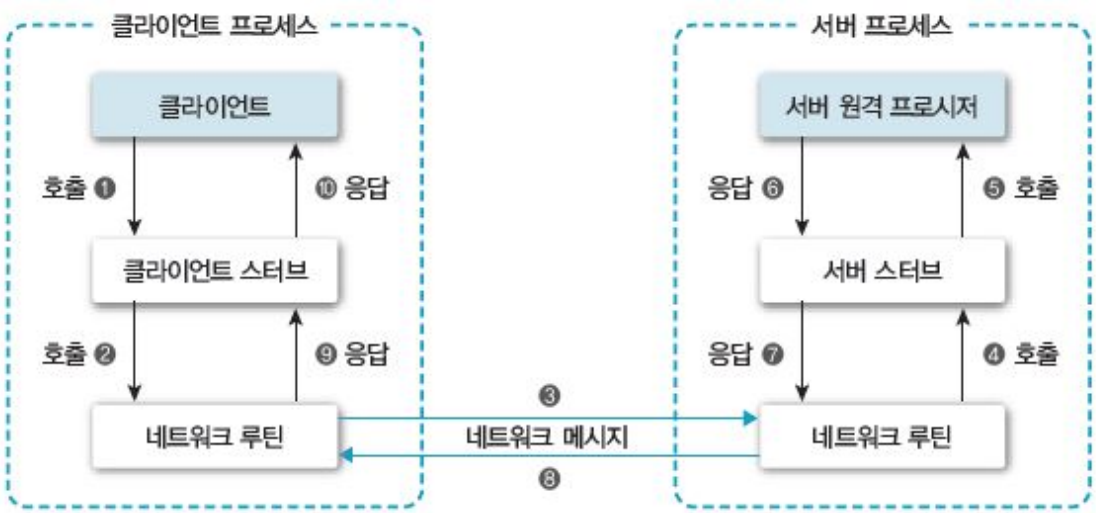


그림 11-8 클라이언트와 서버 간 RPC 원리

- 호출 프로그램과 피호출 프로그램이 데이터 공유하여 호출자와 피호출자 사이의 정보 인자로 전달
- 원격지 컴퓨터의 프로그램을 호출할 때 클라이언트에 필요한 데이터가 원격지에 있다면 프로시저 호출 방법 사용 가능
  - 스터브stub : 전송 데이터를 준비하고 수신 데이터를 변환해서 올바르게 해석할 수 있도록 지원하여 처리 결과를 교환하는 모듈
    - » RPC나 지역 호출이 같도록 만드는데, 클라이언트 스터브와 서버 스터브로 분류
    - » 스터브 프로그램은 데이터를 전송하려고 다양한 형태의 데이터로 서로 변환하는 기능 수행

4. 원격 프로시저 호출RPC, Remote Procedure Call

- RPC 동작 과정



❶ 클라이언트가 클라이언트 스터브를 호출한다. 이때 원격 프로시저 호출을 하는 인자들이 전달되고 이렇게 원격 프로시저에 인자를 포함하는 과정을 마샬링marshalling이라고 한다.

❷ 클라이언트 스터브가 네트워크 메시지를 원격 시스템에 보내려고 운영체제 커널의 네트워크 루틴을 호출한다.

❸ 원격 시스템 커널이 프로토콜로 네트워크 메시지(요청 메시지)를 서버 스터브에 전달한다.

❹ 메시지를 수신한 서버 스터브가 요청 메시지를 프로시저의 인자로 마샬링하고 해당 프로시저를 호출한다.

❺ 서버 스터브가 지역 프로시저 호출(서버 기능 호출)을 실행하여 클라이언트에서 받은 인자를 전달한다.

❻ 프로시저를 완료하면 응답(반환 값)과 함께 서버 스터브에 전달한다.

❼ 서버 스터브는 필요에 따라 응답 값을 표준 형식으로 변환하고, 클라이언트 스터브로 전송하려고 하나 이상의 네트워크 메시지로 마샬링한다.

❽ 클라이언트 스터브는 네트워크로 네트워크 메시지(응답 메시지)를 전송받는다.

❾ 응답 메시지를 다시 클라이언트 스터브에 전달하여 결과 값으로 변환한다.

❿ 결과 값을 요청한 클라이언트에 전달한다.

그림 11-9 RPC(원격 프로시저 호출)의 동작 과정

5. 분산 시스템의 구조와 구축 목적

■ 분산 시스템의 구조

- 저렴한 노드 여러 개를 운영체제 하나가 제어할 수 있도록 구현, 강력한 시스템 구성
- 네트워크로 연결된 여러 노드에 프로그램 하나를 분산하여 실행하면서 마치 하나의 프로그램처럼 동작
- 여러 사용자가 자원 공유하여 대규모 작업 지원하므로 다양한 사용자에게 서비스 가능
- 각 프로세서는 초소형 프로세서부터 워크스테이션, 소형컴퓨터, 대형컴퓨터까지 다양
- 구조

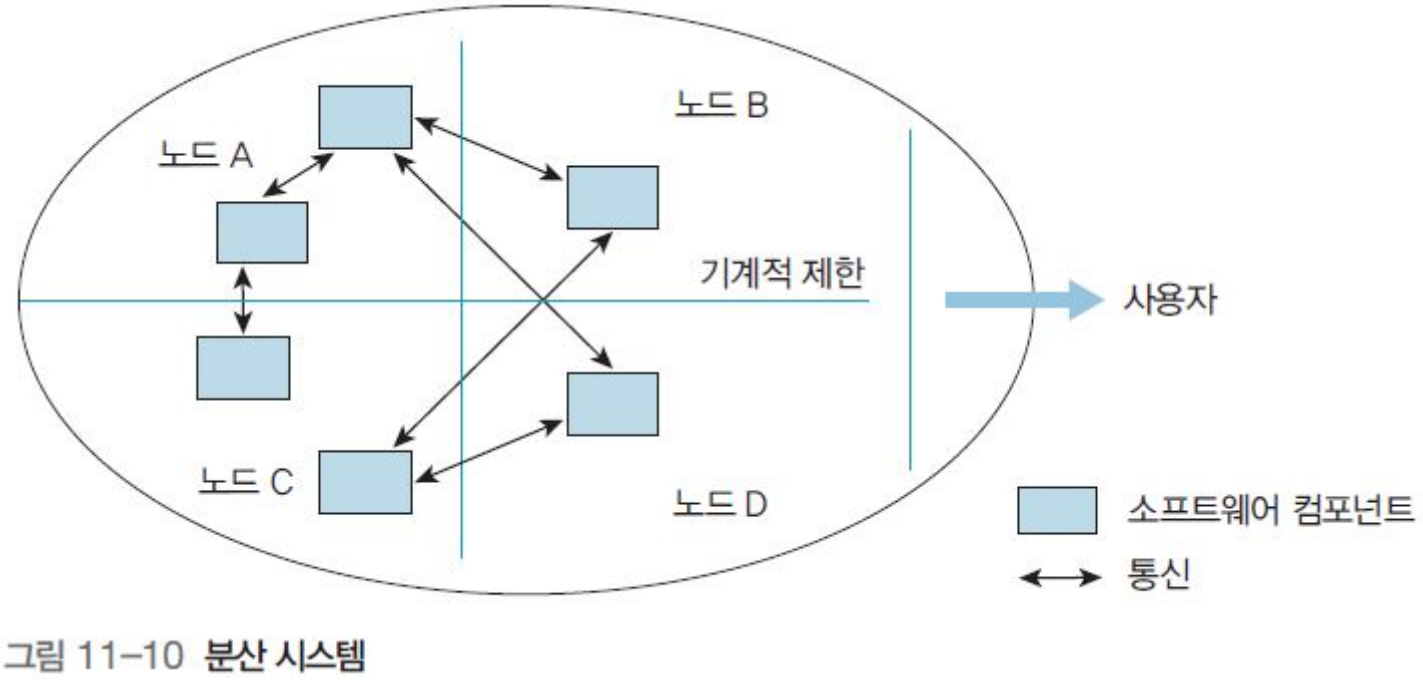


그림 11-10 분산 시스템

5. 분산 시스템의 구조와 구축 목적

분산 시스템의 구축 목적

- 자원 공유 용이
- 연산 속도 향상
- 신뢰성 향상
- 통신 가능

5. 분산 시스템의 구조와 구축 목적

분산 처리 시스템의 기본 목표

- 각종 자원의 투명성transparency 보장
  - 투명성은 상호 연결된 컴퓨터를 사용자가 하나의 컴퓨터 시스템으로 인식할 수 있도록 분산을 감추어 사용자가 이 정보를 몰라도 작업을 수행할 수 있도록 지원하는 것
  - 액세스access 투명성(예 : SQL 쿼리, 웹 내비게이션)
  - 위치location 투명성(예 : 웹페이지, NFSNetwork File System(네트워크 파일 시스템))
  - 고장failure 투명성(예 : 데이터베이스 관리 시스템database management system)
  - 중복replication 투명성(예 : 분산 DBMS, 웹페이지 미러링mirroring)
  - 이동migration(이주) 투명성(예 : 웹페이지, NFS)
  - 영속permanence 투명성
  - 자원resource 투명성
  - 트랜잭션transaction 투명성
  - 재배치reassignment 투명성
  - 규모scale 투명성
  - 병행concurrency 투명성

Section 02 네트워크 운영체제(1. 네트워크 운영체제)

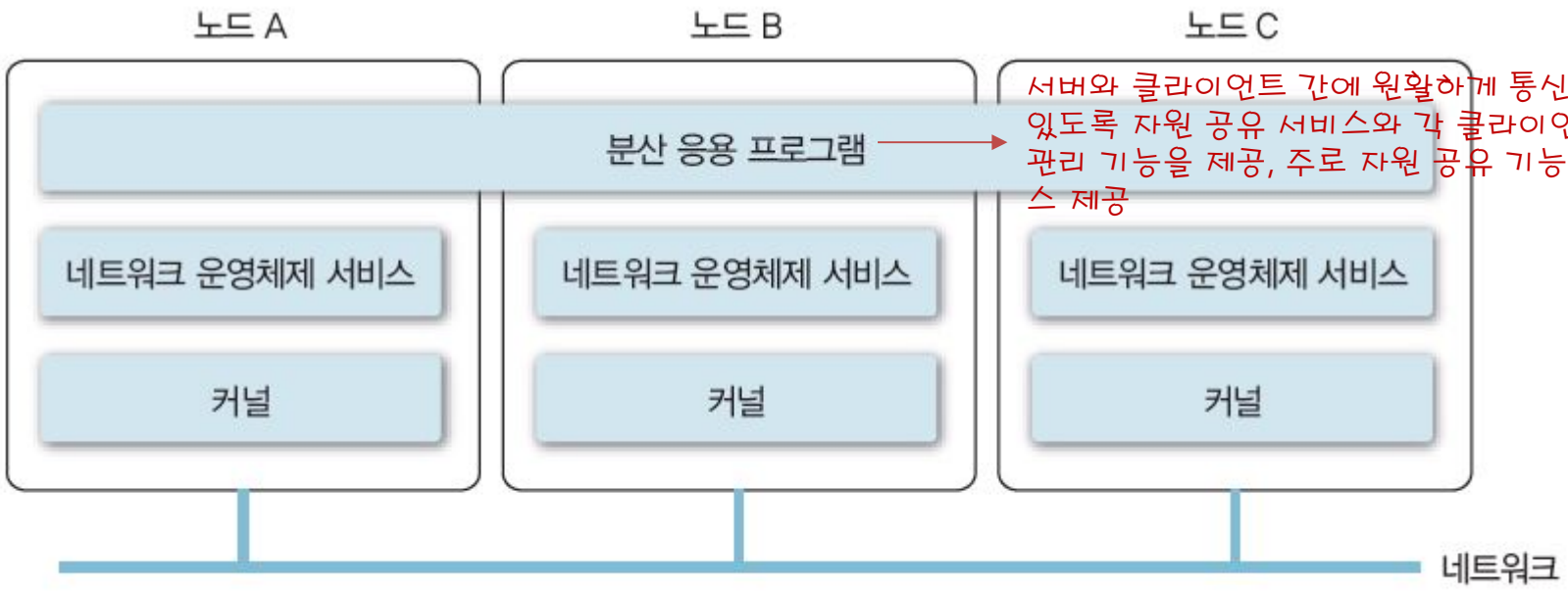
분산 시스템에 사용되는 운영체제

- 네트워크 운영체제
  - 각 컴퓨터가 독자적인 운영체제를 가진 채 사용자 프로그램을 통해 분산 시스템이 구현된 것
  - 낮은 수준의 분산 시스템 운영체제
  - 기기마다 운영체제가 다름
  - 지역적으로 널리 분산된 대규모 네트워크에서 사용하기 때문에 사용자가 기기 및 운영체제의 종류와 사용법을 알고 있어야 함
- 분산 운영체제
  - 시스템 내에 하나의 운영체제가 존재하고, 전체 네트워크를 통틀어서 단일 운영체제로 운영
  - 전체 시스템을 일관성 있게 설계할 수 있음

Section 02 네트워크 운영체제(1. 네트워크 운영체제)

■ 네트워크 운영체제NOS, Network Operating System의 원리

- 통신 제어, 분산된 자원을 공유하면서 독립된 시스템들을 서로 연결하려고 개발
- 일반적인 구조



서버와 클라이언트 간에 원활하게 통신할 수 있도록 자원 공유 서비스와 각 클라이언트의 관리 기능을 제공, 주로 자원 공유 기능 서비스 제공

그림 11-11 네트워크 운영체제의 일반 구조

서버에 저장된 대용량 파일이나 서버에 연결된 공유 프린터 등 자원을 요청한 클라이언트에 제공하려고 설계
자원 공유(프린트 서비스, 주변장치 공유), 파일 전송, 액세스 권한(원격 처리, 사용자 관리), 데이터 보호(보안 인증, 권한 부여), 관리 제어(전자 우편, 네트워크 관리) 등 기능 제공
서버의 분산 응용 프로그램은 서버와 클라이언트 간에 원활하게 통신할 수 있도록 자원 공유 서비스와 각 클라이언트의 관리 기능을 제공, 주로 자원 공유 기능 서비스 제공

1. 네트워크 운영체제

네트워크 운영체제의 종류

- 윈도우 NT 서버 : 윈도우 기반의 1세대 네트워크 운영체제
- 윈도우 2000~2003 서버 : 윈도우 NT가 발전된 네트워크 운영체제
- 노벨 넷웨어Novell Netware : 노벨사가 클라이언트/서버 환경 위해 설계한 네트워크 운영체제
- 유닉스
- 리눅스

1. 네트워크 운영체제

- 네트워크 운영체제의 클라이언트와 서버

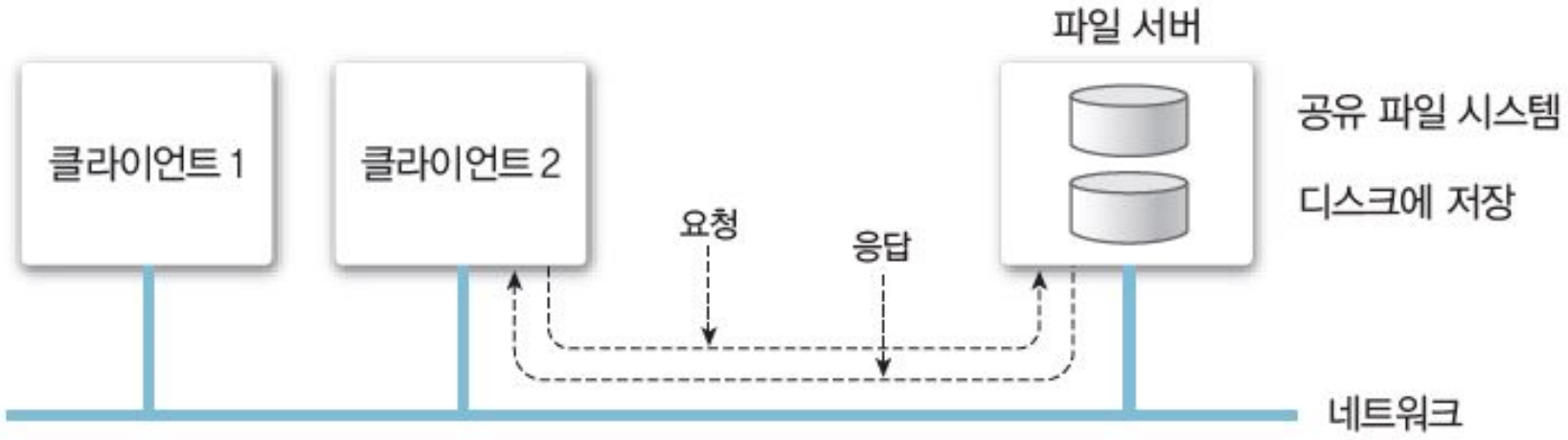


그림 11-12 네트워크 운영체제의 클라이언트와 서버

1. 네트워크 운영체제

네트워크 운영체제의 주요 기능

- 자원 공유 : 네트워크에 연결된 서버와 클라이언트 간 하드디스크나 프린터 등 자원
- 액세스 권한 부여 : 사용자는 원격 사이트의 자원을 사용할 수 있도록 액세스 가능해야 함
  - 자격 있는 사용자만 특정 자원을 사용하도록 제한 가능
  - 호스트 컴퓨터의 하드웨어 사항을 몰라도 되지만, 원격 시스템의 명령은 알아야 함
- 파일 전송 : 한 컴퓨터에서 다른 컴퓨터로 데이터 전송
- 데이터 보호 : 사용자별 적합한 권한(읽기 권한, 쓰기 권한 등) 설정하고 데이터 관리·보호하여 서버에 접근하는 클라이언트 사용자가 서버를 사용할 권한이 있는지 인증
- 관리 제어 : 각 클라이언트의 네트워크 이용 정보와 네트워크에서 발생할 수 있는 여러 가지 문제 해결하고 조절하는 관리 기능 제공

1. 네트워크 운영체제

■ 네트워크 운영체제의 운용 방법

- 네트워크 운영체제는 자원 운영 방법에 따라
  - 피투피peer-to-peer 모델과 클라이언트/서버 모델로 구분

자원은 피투피
네트워크에서 균일하게 공유

자원은 클라이언트/서버 네트워크에서
파일 서버로 제어

그림 11-13 **네트워크 운영체제 운용 형태별 자원 관리**

1. 네트워크 운영체제

■ 피투피 방법

- 동등하게 작동하는 LAN에 연결된 각 컴퓨터가 상황에 따라 클라이언트 또는 서버로 동작하는 방법
- 파일 서버나 중앙집중식 관리 자원이 없으며, 모든 컴퓨터가 동일한 액세스 권한과 네트워크에 있는 사용 가능한 자원에 같은 권한 갖음

표 11-1 피투피 방법의 장점과 단점

장점	• 적은 초기 비용 : 클라이언트/서버의 기능을 한 컴퓨터에서 구현한다. • 자원 : 자원의 활용 극대화와 자원을 각각 균일하게 공유한다.
단점	• 분산 : 파일과 응용 프로그램에 중앙 저장소 없이 분산되므로 관리하기가 어렵다. • 보안 : 서버와 클라이언트에 보안을 제공하지 못한다.

1. 네트워크 운영체제

클라이언트/서버 방법

- 네트워크와 독립적이며, 네트워크에 접속하지 않은 컴퓨터 시스템은 자신의 환경에 맞게 변경·관리 가능
- 네트워크에 연결된 컴퓨터가 각각 임무, 즉 클라이언트는 클라이언트, 서버는 서버로서 역할 수행 방법
- 클라이언트/서버 모델이 워크스테이션 모델이면 클라이언트는 예외적으로 동시에 사용자 한 명이 사용 가능
- 중앙에서 관리하는 형태로 응용 프로그램에 전용 파일 서버를 허용
- 파일 서버는 보안과 자원 액세스를 제공하는 시스템의 핵심
- 개별 워크스테이션(클라이언트)은 파일 서버에 있는 사용 가능한 자원 액세스

1. 네트워크 운영체제

- 클라이언트/서버 구조

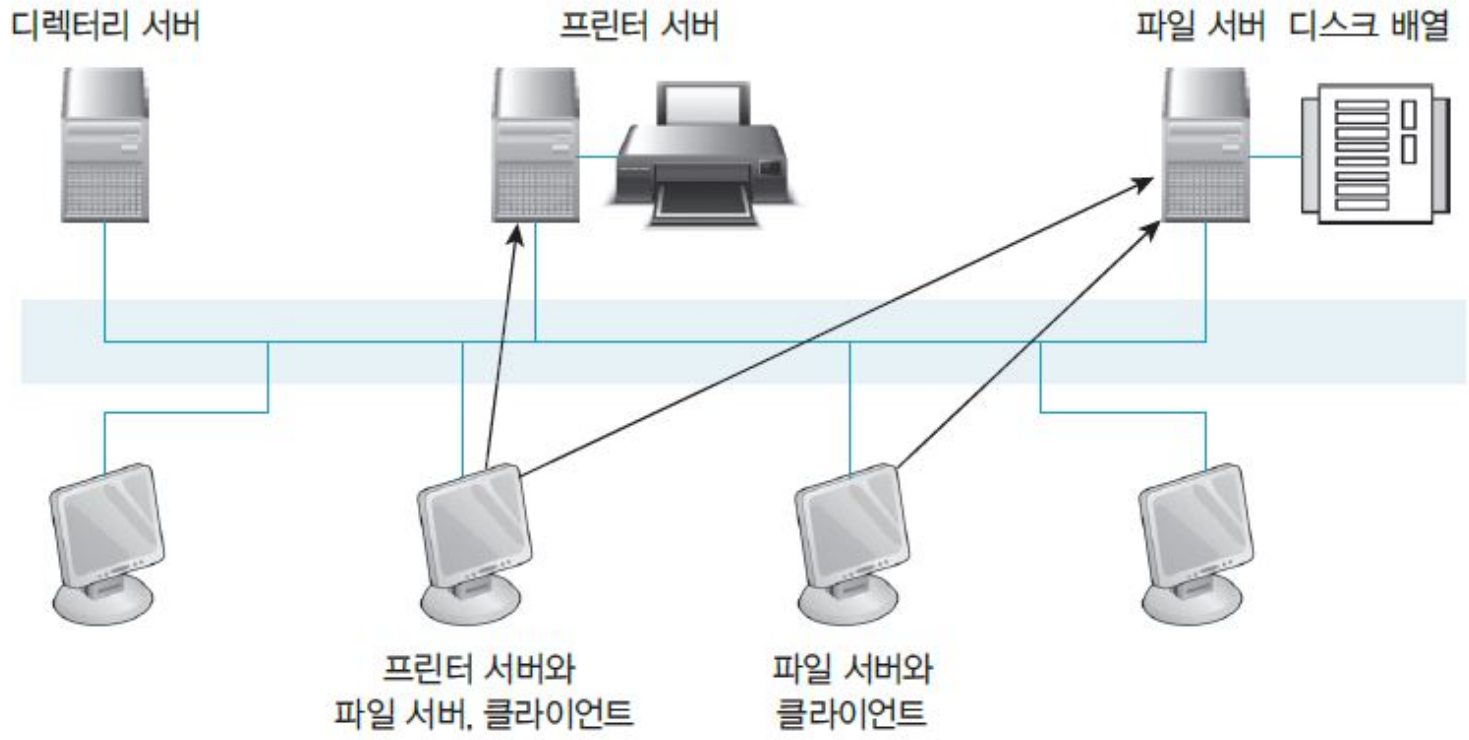


그림 11-15 **클라이언트/서버 구조**

1. 네트워크 운영체제

- 클라이언트/서버 장점과 단점

표 11-2 **클라이언트/서버 방법의 장점과 단점**

장점	• 중앙집중식 : 서버에서 자원과 데이터 보안을 제어한다. • 유연성 : 새로운 기술을 시스템에 쉽게 통합 가능하다. • 접근성 : 서버를 여러 플랫폼이나 원격으로 액세스한다.
단점	• 비용 : 전용 서버에 초기 투자가 필요하다. • 소프트웨어 : 네트워크 운영체제 소프트웨어가 필요하다. • 의존성 : 서버가 다운되면 네트워크가 작업을 중지한다. • 유지 관리 : 대형 네트워크 작업을 효율적으로 동작할 수 있는 관리자가 필요하다.

2. 분산 운영체제DOS, Distributed Operating System의 연산

■ 분산 운영체제의 개념

- 지역적으로 네트워크 운영체제의 자원을 관리, 제어를 제한하는 방법에서 벗어나 시스템 자원을 전역적으로 제어하고 관리할 필요성 느껴서 발전
- 분산 시스템

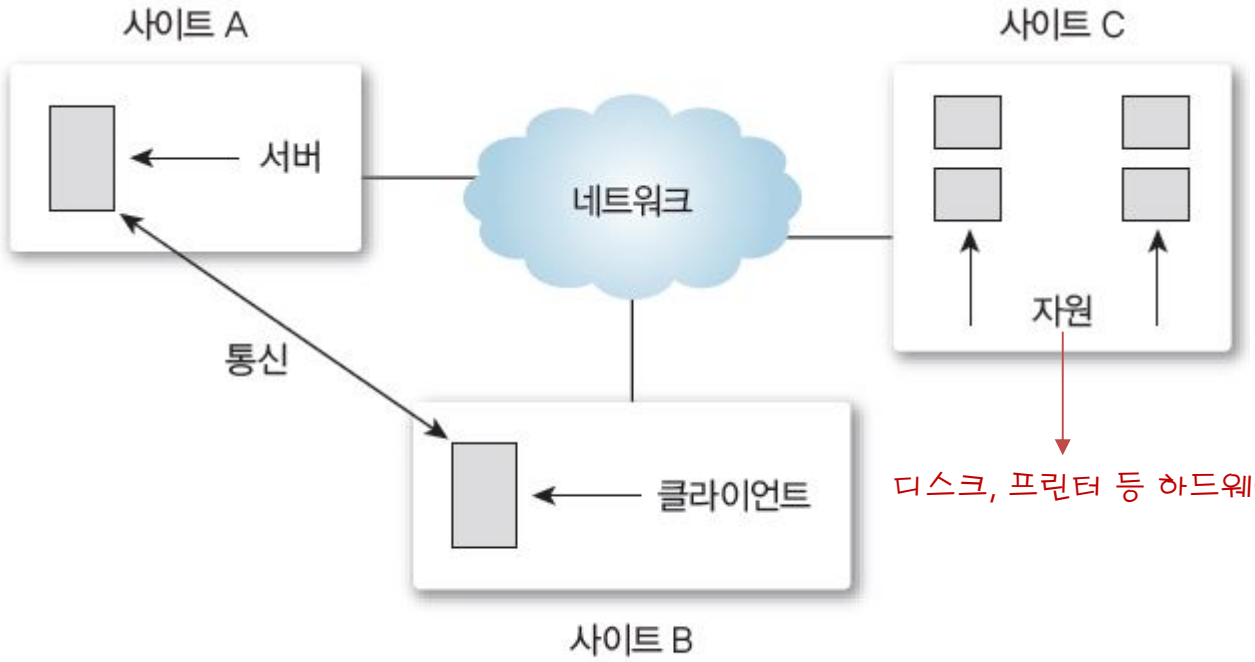


그림 11-16 **분산 시스템**

디스크, 프린터 등 하드웨어와 파일, 프로그램 등 소프트웨어

네트워크가 공유하는 공동 운영체제로 사용자에게 시스템이 제공하는 여러 자원에 액세스할 수 있는 참조 투명성을 제공.
즉, 분산된 컴퓨터 간에 자원을 쉽게 공유하고 액세스할 수 있는 운영체제. 여기서 자원은 디스크, 프린터 등 하드웨어와 파일, 프로그램 등 소프트웨어를 의미한다

2. 분산 운영체제DOS, Distributed Operating System의 연산

■ 분산 운영체제의 제어하에서의 작업 수행

- **데이터 이동**data migration **:** 사이트 A에 있는 사용자가 사이트 B에 있는 파일 등 데이터에 액세스하려면 데이터를 다음 두 가지 방법으로 전송해야 함, 파일의 일부분만 액세스하는 응용 분야에서는 두 번째 방법이 더 유리. 그러나 파일 대부분을 액세스할 때는 파일 전체를 복사하는 것이 더 효율적
  - 파일 전체를 사이트 A로 전송하는 방법. 이때 해당 파일에서는 지역적으로 모두 액세스. 사용자가 더는 파일을 액세스할 필요가 없을 때 수정했을지도 모르나, 파일 복사본은 사이트 B로 되돌려 보냄
  - 실제로 작업에 필요한 부분만 사이트 A로 전송하는 방법. 사용자가 더는 그 파일에 액세스할 필요가 없으면 수정된 파일 일부분만 사이트 B로 되돌려 보냄(요구 페이징 방법과 비슷)

2. 분산 운영체제DOS, Distributed Operating System의 연산

- **연산 이동computation migration**
  - 어떤 환경에서는 데이터 이동이 아닌 연산 이동이 더 효율적
  - 이런 연산 방법으로 프로세스 P가 사이트 A에 있는 파일에 액세스하면 사이트 A에서 파일을 액세스하는데(원거리 프로시저 호출), 프로세스 P는 사이트 A에서 미리 정의된 프로시저를 불러 적절히 수행한 후 인자를 받음(단, 다른 방법으로는 프로세스 P가 사이트 A로 메시지를 보낼 수 있음). 즉, 사이트 A의 운영체제는 미리 지정된 작업을 수행하는 프로세스 Q를 만들고, 프로세스 Q는 수행을 끝낸 후 그 결과를 P에 돌려줌
  - 이런 방법에서 프로세스 P는 프로세스 Q와 동시에 수행할 수 있으며, 사실상 여러 사이트에서 동시에 수행하는 여러 프로세스가 있을 수 있음

2. 분산 운영체제DOS, Distributed Operating System의 연산

- **프로세스 이동process migration**
- 프로세스의 전체나 일부를 다른 사이트에서 실행하는 것의 유리한 점
  - 부하 균등화 : 프로세스(작업)는 네트워크로 작업량을 분산
  - 연산 속도 향상 : 다른 사이트에서 동시에 수행할 수 있는 여러 서브 프로세스로 나눈다면 전체 프로세스의 반환시간 줄어듬
  - 하드웨어 이익 : 어떤 프로세스는 한 특징 프로세서에서 수행에 적합
  - 소프트웨어 이익 : 어떤 특정 사이트에서만 사용할 수 있는 소프트웨어가 필요하거나 소프트웨어는 이동할 수 없으므로 작업을 이동시키는 것이 효율적
- 프로세스 이동 방법의 분류
  - 사용자가 모르게 시스템이 클라이언트에서 프로세스 이동 사실을 숨기는 방법
    - 사용자는 이동을 한다는 외부 명령을 프로그램에 코딩할 필요 없음. 부하 균등화와 연산 속도 향상에 사용
  - 사용자가 직접 프로세스 이동 방법을 명시하는 요구나 허가하는 방법
    - 하드웨어나 소프트웨어의 이점을 얻는 데 사용

3. 분산 운영체제의 구현

프로세스 기반 분산 운영체제

- 시스템의 모든 프로세스와 자원을 하나의 집합, 즉 시스템 서비스를 제공하는 프로세스의 집합으로 해석
- 프로세스 간 동기화와 시스템 상태, 사용자 프로세스의 제어, 프로세스 생성과 스케줄링 등 프로세스 관리는 프로세스 간에 메시지 교환하여 수행
- 인터럽트도 해당하는 프로세스에 메시지를 전송하여 구현 가능
- 모든 자원은 관리인 서버가 제어
- 서버는 각 자원에서 서비스 요청을 받아들인 후 다른 프로세스의 서비스 요청과 함께 공정하게 제공

3. 분산 운영체제의 구현

객체 기반 분산 운영체제

- 시스템을 자원과 프로세스의 구성으로 해석하지 않고 객체의 집합을, 즉 각 종류의 하드웨어와 이것을 관리하는 소프트웨어의 집합을 독립된 객체로 해석
- 객체는 프로세서, 메모리, 프린터, 디스크 등 하드웨어와 파일, 프로그램 데이터 등 소프트웨어로 표현하거나 모두 혼합한 의미로 표현, 시스템에서 각 객체에는 고유한 이름이나 식별자가 있음
- 객체는 상태가 변화하며 정해진 패턴에 따름
- 프로세스 관리는 독립된 객체로 동작하는 프로세스의 객체 관리가 됨, 이때 프로세스 관리는 시스템에서 객체의 운영과 생성, 제거를 제어하는 커널 수준과 프로세스 관리자 요소가 필요
- 커널은 동적으로 객체를 생성하고 삭제하는 능력과 함께, 객체를 생성하면 그 동작에 필요한 모든 자원을 할당받아 작업이 끝날 때까지 제어하는 권한 갖음
- 작업이 끝나면 다음 실행을 결정하는 커널에 제어권을 넘겨줌

4. 분산 시스템에서 프로세스 관리

■ 자원 할당 교착 상태

▪ 교착 상태 예방

- 교착 상태가 발생하는 네 가지 필요조건(상호 배제, 점유와 대기, 비선점, 순환 대기 조건) 중 하나만 방지해도 분산 교착 상태 예방 가능.
- 교착 상태는 순환 대기 상태를 탐지하고, 탐지되면 이것을 복구하여 해결.
- 순환 대기 조건은 자원의 종류에 따라 선행적인 순서를 정의.
- 자원을 요청한 순서가 다르면 자원을 점유하는 상태 발생. 또 다른 조건인 점유와 대기 조건이 시스템에서 발생하지 않으려면 작업을 수행하기 전에 필요한 모든 자원을 각 프로세스가 한꺼번에 요구하고 동시에 허용할 때까지 프로세스를 보류해야 하지만 자원의 낭비가 심하고 긴 시간 동안 지연될 수 있음

4. 분산 시스템에서 프로세스 관리

- 교착 상태 탐지
  - 교착 상태를 탐지하려면 시스템에서 모든 자원의 점유와 요구 사항을 자원 그래프로 표현하여 사이클 여부를 검사
  - 분산 시스템에서 프로세스 간 의존관계는 대기 그래프로 나타낼 수 있음
  - 그래프에서 사이클을 방지하려면 주어진 프로세서가 실행에 필요한 자원을 확보할 때까지 실행 지연
- 시스템 제어를 통한 교착상태 탐지 방법
  - 중앙집중형 제어 : 노드 하나가 교착 상태 탐지 전담. 이때 모든 요구와 양도 메시지를 특정 자원을 제어하는 중앙 프로세서로 전송하면, 시스템의 상태를 모두 파악할 수 있으므로 교착 상태를 탐지
    - 구현하기가 쉬우며, 비교적 작고 통신 속도가 높은L AN에 효율적
    - 자원 요청이 중앙 프로세서에 집중되어 병목 현상 발생
    - 중앙프로세서에 문제가 생기면 시스템의 노드가 모두 자원 얻을 수 없음
  - 계층형 제어 : 노드를 계층 구조로 구성하고 한 노드가 루트 역할을 담당하게 한 후 종속된 노드들의 자원 할당 정보를 수집하여 교착 상태 탐지
  - 분산형 제어 : 모든 프로세스가 협력하여 교착 상태 탐지
    - 정보를 먼저 교환해야 하기에 상당한 오버헤드 발생
    - 한 노드에 오류가 발생해도 다른 노드가 적절히 대응할 수 있으므로 장애 내구성 높음

4. 분산 시스템에서 프로세스 관리

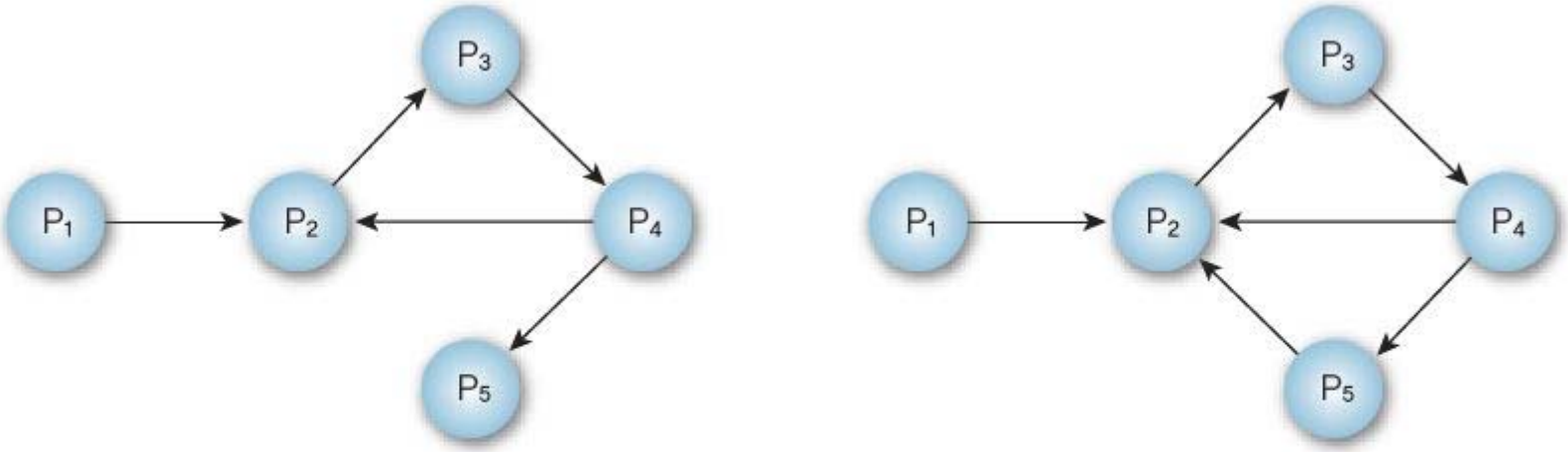
- 교착 상태 회피
  - 분산 시스템에서 교착 상태 회피는 자원 할당 요구를 허용한 후 교착 상태 여부를 결정해야 하기 때문에 매우 곤란. 특히 모든 노드가 시스템의 전역상태를 파악하고 유지해야 하는데, 저장소와 과도한 통신 오버헤드 발생. 안전한 상태를 검사하는 것도 상당한 처리 오버헤드 야기. 시스템은 실행하기 시작하면 다른 장치를 요구하지 않고 완료할 수 있는 트랜잭션에서만 실행 허용해야 함

- 교착 상태 복구
  - 다른 프로세스에 필요한 자원을 많이 사용하는 프로세스를 선택하여 제거한 후 다시 처음부터 재실행하도록 조치. 그리고 제거한 프로세스의 자원은 다른 프로세스에 할당하여 실행

4. 분산 시스템에서 프로세스 관리

■ 메시지 전송 교착 상태

- 프로세스가 메시지를 기다리는 상태로, 상호 대기와 메시지 버퍼 미적용으로 발생
- 상호 대기
  - 동일한 그룹에서 한 프로세스가 다른 프로세스가 보낸 메시지를 기다리고 있고, 전송 중인 메시지가 없을 때 발생.
  - 시스템의 교착 상태는 WFG[wait for graph]라는 대기 그래프로 표현

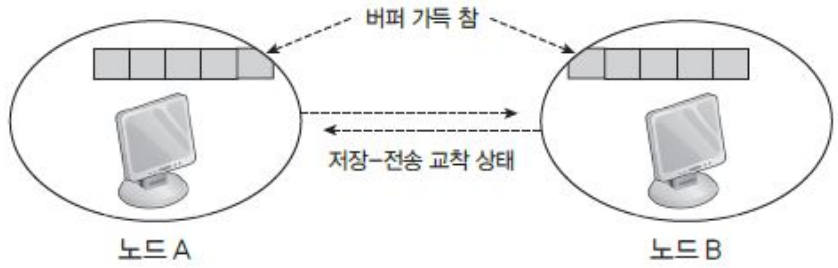


(a) 교착 상태 아님 (b) 교착 상태

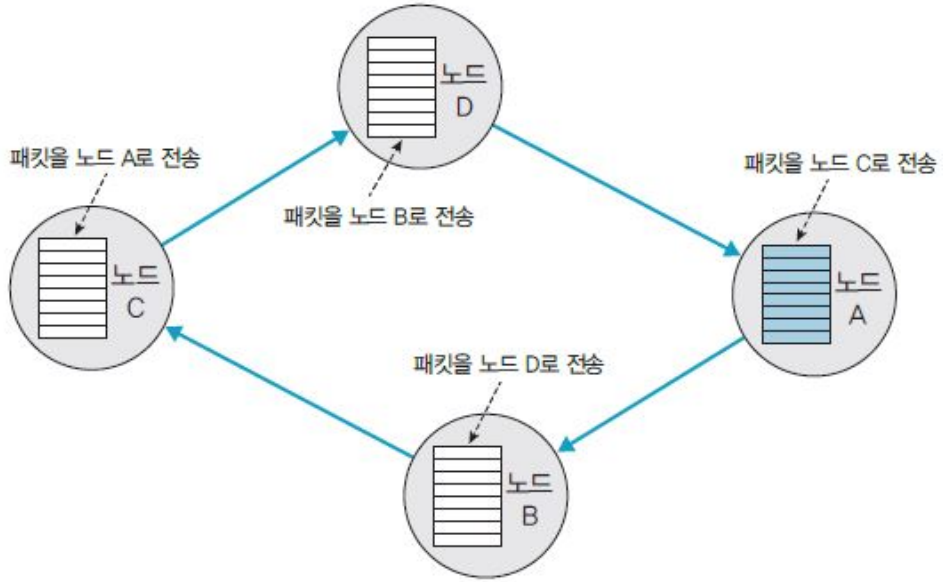
그림 11-17 상호 대기로 발생한 교착 상태

4. 분산 시스템에서 프로세스 관리

- 메시지 할당 버퍼 불충분
  - 메시지 전송 과정에서 교착 상태는 전송 중인 메시지를 저장하는 버퍼 할당과 관련 있음



(a) 메시지 직접 저장-전송 교착 상태



(b) 메시지 간접 저장-전송 교착 상태

그림 11-18 **메시지 버퍼 미적용으로 발생한 교착 상태**

각 링크에 별도의 버퍼를 하나씩 사용하게 하여 예방 가능
공동 버퍼를 사용해도 특정 링크가 버퍼 공간을 모두 사용하지 않는다면 교착 상태는 발생 않으므로 시스템의 각 프로세스 간에 전송할 수 있는 메시지 수에 상한 값을 부여하여 필요한 버퍼를 할당하면 교착 상태 예방 가능
다른 방법은 계층화된 버퍼 풀을 사용하여 교착 상태를 제거

5. 클라이언트/서버 분산 컴퓨팅

■ 클라이언트/서버 시스템의 정의

- 분산 처리 환경 대표적인 예

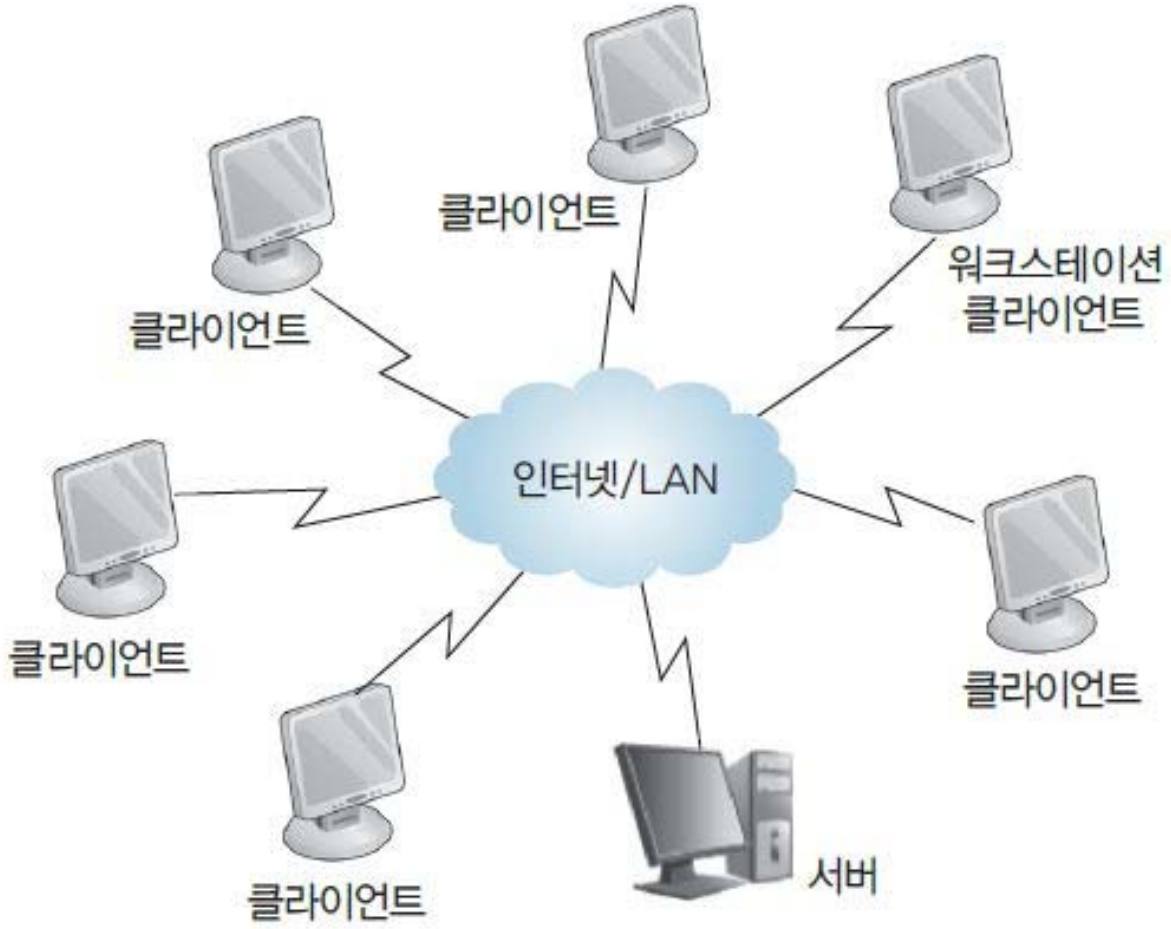


그림 11-19 클라이언트/서버 분산 컴퓨팅 환경 : 클라이언트, 서버, 통신 네트워크로 구성

5. 클라이언트/서버 분산 컴퓨팅

- 클라이언트/서버의 분산 컴퓨팅 환경에서는 사용자, 응용 프로그램, 자원들을 네트워크(단일 LAN이나 WAN)로 서로 연결
- 서버에 있는 응용 프로그램 하나를 독립 된 역할을 하는 다수의 작업으로 분할하여 클라이언트의 서비스 요청 처리

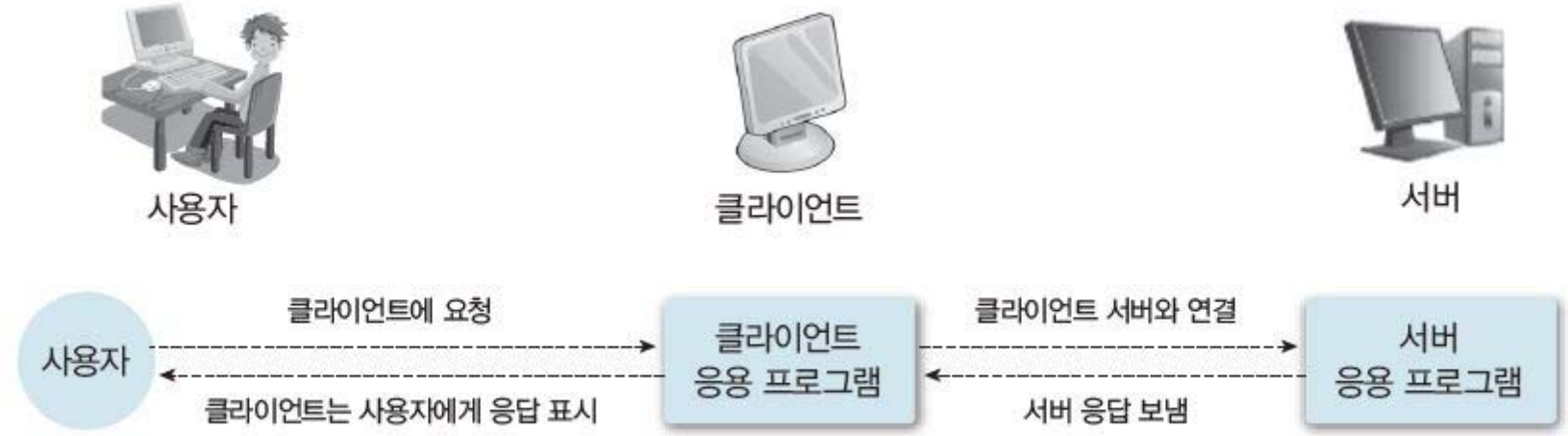


그림 11-20 **클라이언트/서버 컴퓨팅**

- 클라이언트/서버 개념을 단일 컴퓨터에도 적용할 수는 있지만, 네트워크 환경에서 의미가 더 큼
- 네트워크에서 클라이언트/서버는 여러 지역에 걸쳐 분산된 프로그램을 서로 연결시켜주는 편리한 수단 제공(대표적인 클라이언트 : 웹 브라우저)

5. 클라이언트/서버 분산 컴퓨팅

2계층의 일반적인 클라이언트/서버 구조

- 클라이언트/서버 시스템을 구축할 때 핵심 요소는 사용자와 시스템 간에 원활한 클라이언트 컴퓨터에서는 사용자 인터페이스 설계 매우 중요
  - 사용하기 쉬우면서도 친밀한 인터페이스와 강력하고 유연한 GUI 제공
- 클라이언트/서버는 2계층이나 3계층으로 구현
- 클라이언트와 서버가 물리적으로 서로 독립된 시스템에 존재하는 형태로 구성되어 클라이언트와 서버의 플랫폼과 운영체제는 서로 다를 수 있음
- 클라이언트와 서버를 상호 연동할 수 있게 하는 통신 소프트웨어(OSI 참조 모델과 TCP/IP 등)와 분산 응용 프로그램의 기반을 제공하므로(동일한 응용 프로그램 지원) 공유 자원, 작업을 클라이언트와 서버 간에 분할 가능

OSI와 TCP/IP
ISO International Standards Organization의 OSI 참조 모델Open System Interconnection reference model은 이기종 시스템 간에 네트워크를 연결하려고 제안되었다. 이 모델은 네트워크를 계층 7개로 나눠서 네트워크 매체를 이용하여 응용 프로그램의 정보를 다른 컴퓨터의 응용 프로그램에 어떻게 전달하는지 설명
TCP/IP Transmission Control Protocol/Internet Protocol는 현재 가장 널리 사용하는 네트워크 계층 프로토콜
처음에는 프로토콜의 집합 중에서 인터넷 핵심 기능을 제공하는 TCP와 IP만을 지칭했으나, 현재는 TCP/IP 네트워크를 구성하는 데 필요한 기능을 제공하는 관련 프로토콜의 집합을 총체적으로 지칭

5. 클라이언트/서버 분산 컴퓨팅

- 2계층의 클라이언트/서버 구조 : 일반적인 구조

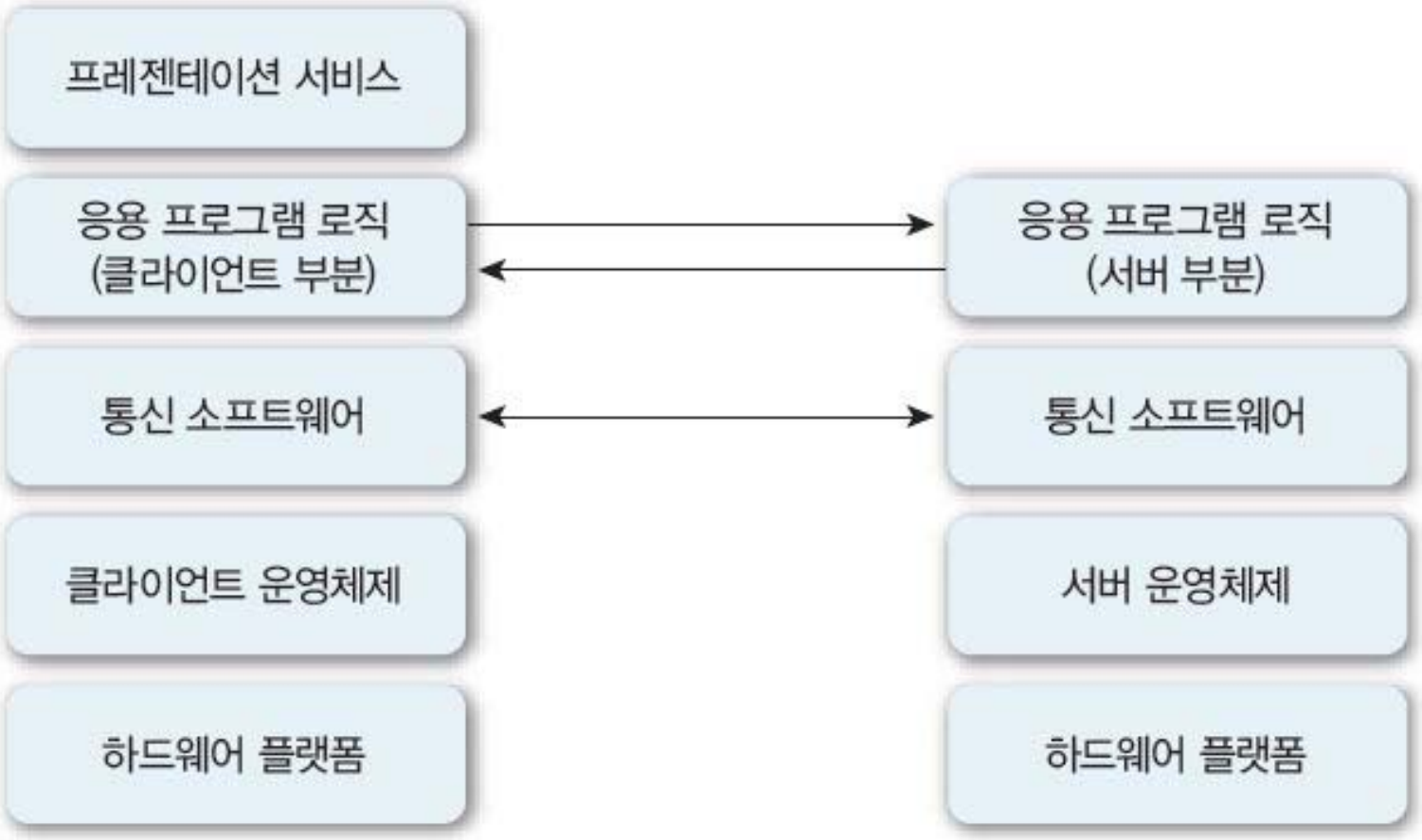


그림 11-21 2계층의 클라이언트/서버 구조 : 일반적인 구조

5. 클라이언트/서버 분산 컴퓨팅

- 클라이언트/서버 시스템에서 응용 프로그램 처리 모듈의 예

그림 11-22 **팻 클라이언트와 씬 클라이언트**

5. 클라이언트/서버 분산 컴퓨팅

클라이언트/서버 시스템에서 응용 프로그램 처리 모듈

- 팻fat 클라이언트
  - 클라이언트 기반 처리 방법. 최적으로 수행할 수 있는 데이터 확인 루틴과 데이터베이스 관련 로직 기능
  - 실질적으로 모든 응용 프로그램을 처리할 수 있도록 응용 프로그램의 처리 모듈과 프레젠테이션 로직이 클라이언트에 위치. 작성한 SQLStructured Query Language문을 원격지 컴퓨터에 위치한 데이터베이스 서버에 전달하면, 서버가 데이터베이스 읽기/쓰기만 처리한 결과를 다시 클라이언트에 보내는 데이터 전송 형태의 원격 데이터베이스 액세스 모델
- 씬thin 클라이언트
  - 서버에는 응용 프로그램 처리 모듈에 해당하는 저장 프로시저와 데이터베이스가 위치하고, 클라이언트에는 프레젠테이션 로직만 위치하는 구조
  - 클라이언트는 단순히 서버의 프로시저를 호출하는 기능으로 사용자 요구 처리

5. 클라이언트/서버 분산 컴퓨팅

2계층 모델의 문제점

- 클라이언트/서버 간에 네트워크 통신량 증가하여 시스템의 성능 떨어지고 네트워크 병목 현상이 발생. 파일 서버를 사용할 때, 파일 입출력은 네트워크에서 발생하는 지연 때문에 지역 파일 참조에 비해 성능 현저히 저하. 파일 캐시를 사용하여 원격 서버의 접근 횟수 감소시킬 수 있으나, 캐시의 일관성 문제를 해결해야 함
- 응용 프로그램의 논리적 구조와 물리적 구조를 완전하게 분리하지 않아 업무 프로세스나 업무환경을 변경할 때마다 많은 부분의 소프트웨어를 수정해야 함
- 대부분의 응용 프로그램이 특정 데이터베이스에 종속되어 데이터나 응용 프로그램이 증가하면 통합 어렵고 유연성 떨어짐)확장성의 한계). 그리고 개발된 응용 프로그램의 이식성 한계로 한 서버에서 다른 서버로 프로그램 기능의 일부 이동 곤란

5. 클라이언트/서버 분산 컴퓨팅

3계층의 클라이언트/서버 구조

- 2계층의 클라이언트/서버 구조 문제점 해결 방안
- 데이터베이스가 위치한 서버 부분과 사용자가 주로 사용하는 클라이언트를 완전히 분리하고, 응용 프로그램 로직 모듈화해서 중간 계층에 별도로 두어 유연하고 확장 가능한 시스템
- 파일 캐시의 일관성을 유지할 수 있는 액세스 방법. 네트워크와 운영체제의 불일치를 극복할 수 있는 계층으로, 미들웨어를 포함하여 분산 클라이언트/서버 컴퓨팅 실현

그림 11-23 3계층의 클라이언트/서버 구조

5. 클라이언트/서버 분산 컴퓨팅

3계층 모델의 주요 장점

- 클라이언트의 작업 처리 부담을 덜고 응용 프로그램 서버가 작업 처리를 맡으므로 중간 계층은 프로세스 대기, 응용 프로그램 실행 등 많은 기능을 실행할 수 있어 시스템의 성능과 융통성 많이 향상
- 응용 프로그램 서버에 작업 로직을 배치하여 프로그래밍 언어와 관계없이 개발 가능, 응용 프로그램 로직을 모듈화하는 방법으로 유지 관리 용이
- 확장성 크게 향상. 데이터베이스 서버는 응용 프로그램 서버 기능을 하지 않고 데이터 작업만 수행하여 데이터베이스 서버의 처리 용량 증가

5. 클라이언트/서버 분산 컴퓨팅

클라이언트/서버 구조와 미들웨어

- 미들웨어middleware : 클라이언트/서버 사이에서 교량 역할, 서비스 제공하는 소프트웨어
  - 서로 다른 시스템 간에 상호 운영을 하는 데 필요한 소프트웨어, 즉 네트워크, 데이터베이스, 운영체제에서 호환성을 제공한다. 네트워크가 기본으로 구축된 분산 컴퓨팅 환경에서 사용자 컴퓨터와 네트워크에서 실행되는 응용 프로그램 간에 자유롭게 데이터를 이동하여 응용 프로그램 개발을 지원하는 소프트웨어
  - 3계층 클라이언트/서버 구조에서는 필수적인 기술
  - 미들웨어의 가장 큰 역할은 클라이언트와 서버 간 데이터 교환
  - 미들웨어의 기본 목표는 클라이언트에 있는 응용 프로그램이나 사용자가 서버 종류에 관계없이 각 서버에 있는 다양한 서비스 이용할 수 있는 시스템 구축
  - 단일 시스템 컴퓨팅처럼 쉽게 분산 컴퓨팅을 구현할 수 있도록 응용 프로그램의 구성 요소를 서로 결합하면서 클라이언트와 서버를 연결하는 수단 제공
  - 운영체제와 응용 프로그램 사이에 위치

5. 클라이언트/서버 분산 컴퓨팅

- 클라이언트/서버 구조에서 미들웨어의 위치와 역할

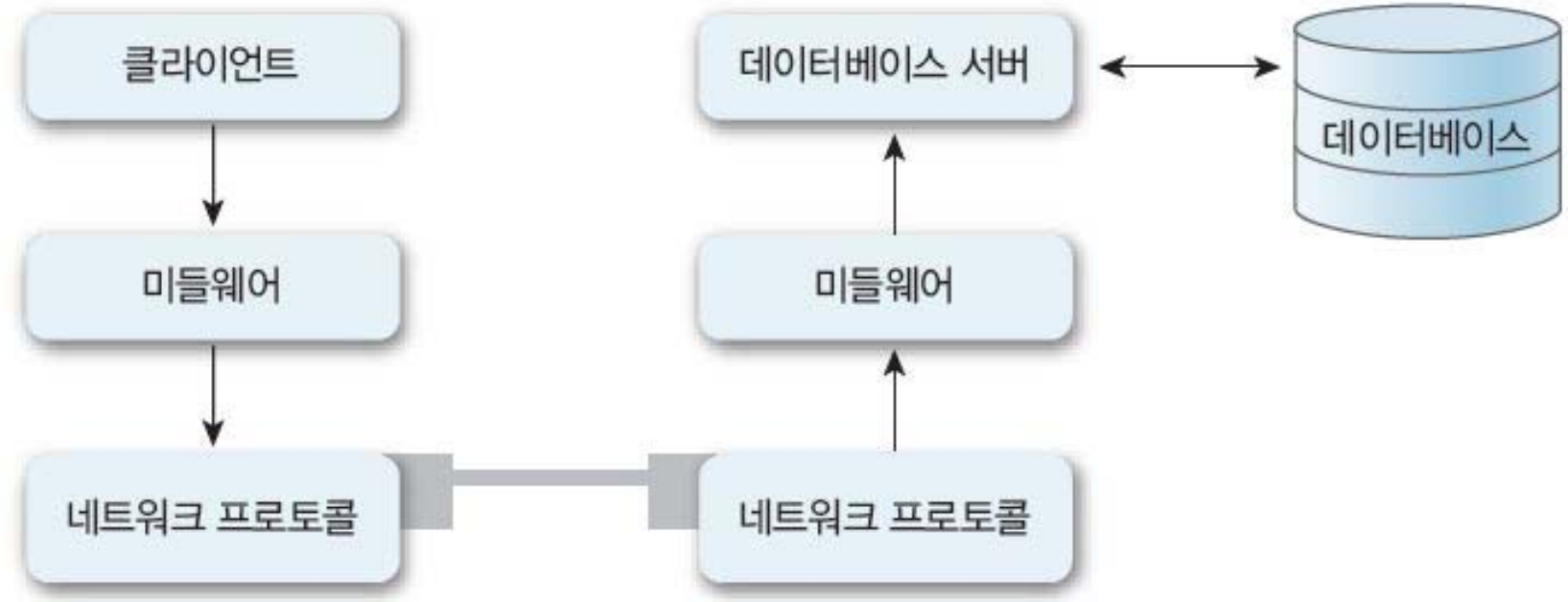


그림 11-24 클라이언트/서버 구조에서 미들웨어의 위치와 역할

5. 클라이언트/서버 분산 컴퓨팅

- 미들웨어를 이용한 분산 시스템의 일반적 구조

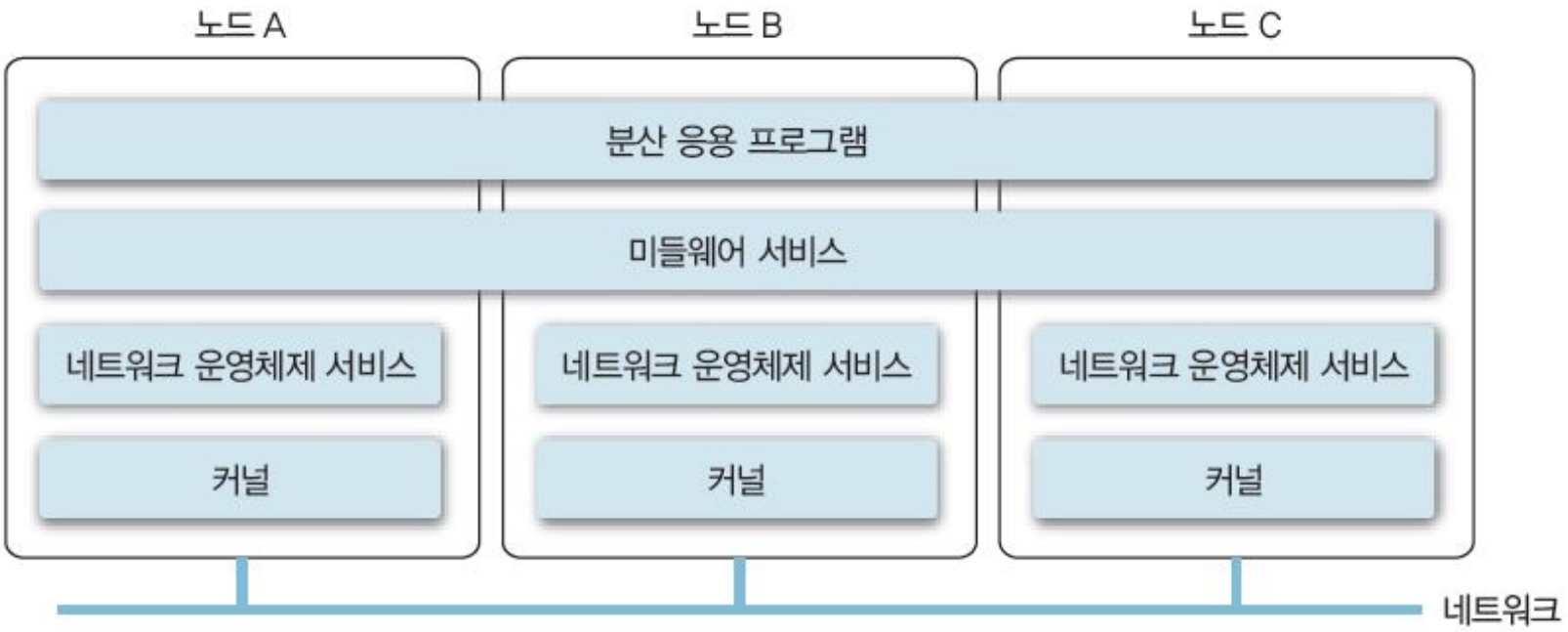


그림 11-25 **미들웨어를 이용한 분산 시스템의 일반적 구조**

5. 클라이언트/서버 분산 컴퓨팅

- 미들웨어의 구성

그림 11-26 **미들웨어의 구성**

Section 03 다중 처리 운영체제 (1. 구조와 원리)

다중 처리multiprocessing 시스템의 구조와 원리

- 병렬 처리라고도 함
- 다수의 프로세서를 동시에 수행하여 시스템성능을 향상시키는 방법
- 다중 처리 시스템은 단위시간당 처리하는 양을 늘리는 것이 목적이므로, 각종 연산을 병렬 수행하려면 프로세서가 매우 많아야 함. 그래서 병렬성parallelism 방법을 이용하면 주어진 프로그램을 가장 짧은 시간 내에 마칠 수 있으나, 대부분의 프로그램은 순차적으로 실행
- 병렬성 프로그램 작성이 어려운 이유
  - 인간의 사고는 병렬적으로 생각하기가 어려움
  - 인간의 언어는 병렬성 적절히 표현하지 못함
  - 병렬성 때문에 다중 처리를 만든 것이 아니라서 병렬성 경험 부족
  - 컴퓨터 하드웨어는 순차적 처리에 익숙
  - 병렬 프로그램은 오류 검색 어려움
- 막대한 비용 없이 단일 프로세서 컴퓨터 시스템의 계산 능력을 높일 수 있는 방법 제공
- 시스템을 모듈식으로 설계함으로써 프로세서를 추가하여 전체 시스템의 능력 확장 가능

■ 다중 처리 시스템을 설계할 때 고려할 특징

- 다중 처리 시스템은 성능이 거의 비슷한 2개 이상의 프로세서 포함
- 모든 프로세서는 동일한 메모리 공동으로 사용
- 모든 프로세서는 입출력 채널과 제어장치, 그 외의 장치들 공동 사용
- 전체 시스템은 하나의 운영체제로 운영. 이 운영체제는 프로세스와 각 작업의 상호작용을 여러 단계에서 도와줌. 프로그램, 데이터 집합, 데이터 단위들의 단계에서 상호작용 도움
- 다중 시스템 설계 예

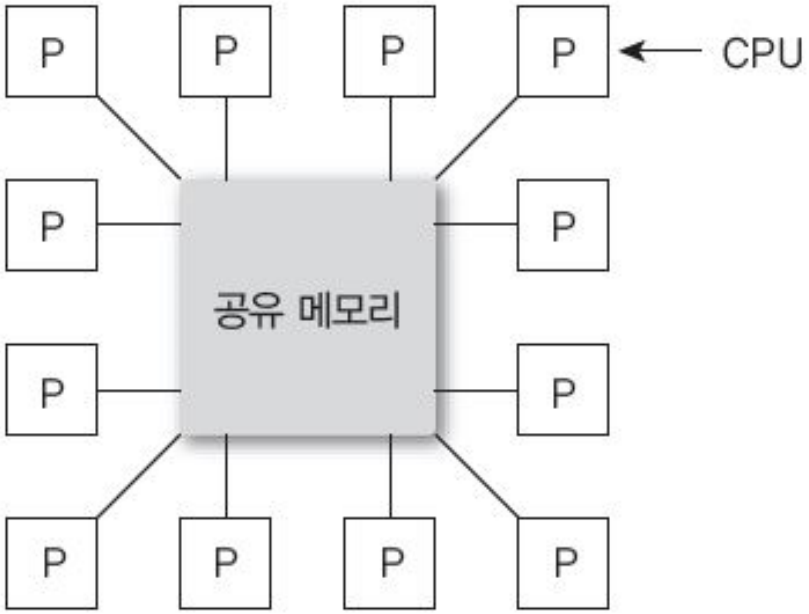


그림 11-27 다중 시스템 설계 예 : 공유 메모리에 여러 프로세서 연결

2. 다중 처리 시스템의 연결 방법

공동 버스common bus 시스템

- 프로세서, 메모리, 입출력장치 등 각종 장치 간에 하나의 공동 버스 제공하는 방법
- 수동장치로, 각종 장치간 정보 전송을 해당 장치의 버스 인터페이스가 제어
- 데이터 전송을 원하는 프로세서는 버스와 상대방 장치를 사용할 수 있는지 검사, 상대방 장치에 데이터 처리 방법 알린 후 데이터 전송해야 함
- 데이터를 전송받는 장치는 버스에 실린 메시지가 자신에게 오는 정보라는 것을 알 수 있어야 하므로 전송장치에서 제어 신호들을 인식하여 따를 수 있어야 함
- 제공해야 하는 기능
  - 어드레싱 : 버스에서 모듈을 구별하여 데이터의 목적지 결정
  - 중재 : 모든 입출력 모듈은 일시적으로 마스터 기능을 수행, 버스 제어는 우선순위 방법을 사용하여 경쟁적 요청을 중재하는 데 제공
  - 시간 공유 : 하나의 모듈이 버스를 제어할 때 다른 모듈은 잠김 및 버스 액세스 달성 때까지 작업 일시 중단
- 문제점
  - 버스에 이상이 생기면 전체 시스템 가동 불가
  - 시스템의 전체 전송량이 버스 전송률에 따라 제한
  - 시스템이 바빠지면 버스를 사용하려고 경쟁하므로 효율 떨어짐

2. 다중 처리 시스템의 연결 방법

- 공동 버스 시스템

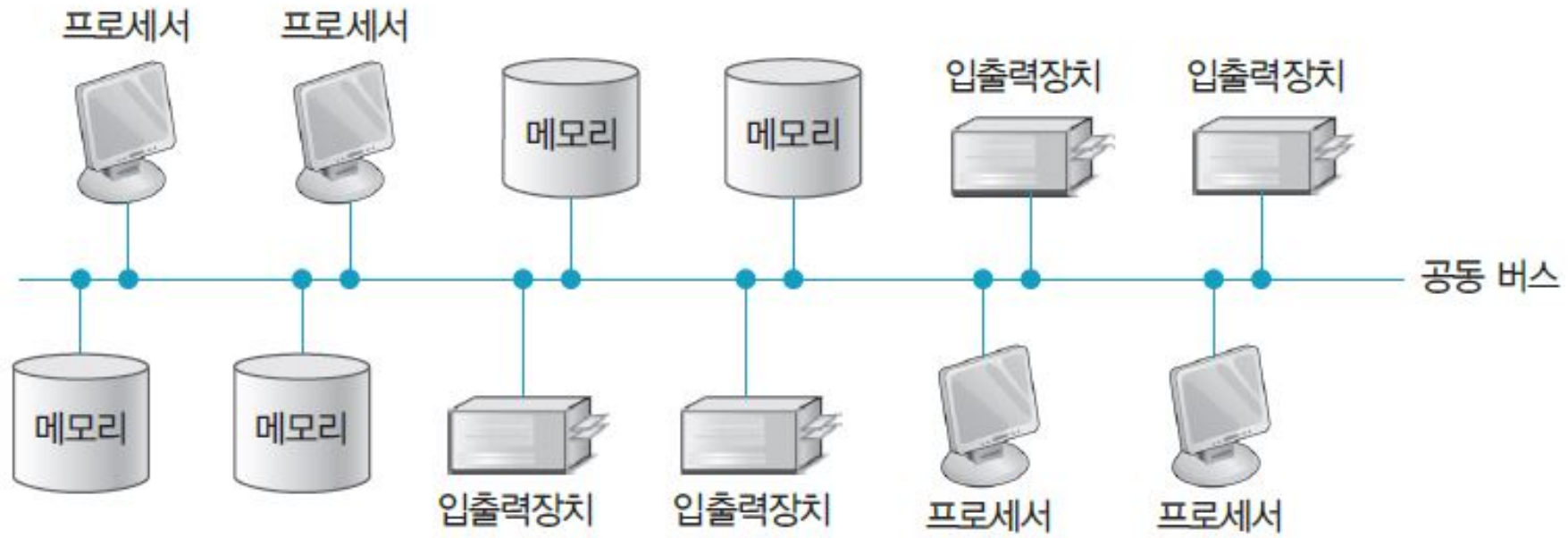


그림 11-28 **공동 버스 시스템**

2. 다중 처리 시스템의 연결 방법

- **크로스바 교환 행렬crossbar switch matrix 시스템**
  - 공동 버스 시스템의 버스 수를 메모리 수만큼 증가
  - 메모리마다 회선이 달라 서로 다른 메모리 2개를 동시에 참조할 수 있고, 충돌 없음
  - 모든 메모리를 동시에 전송 가능
  - 하드웨어가 매우 복잡할 수 있는데, 교환기가 복잡해지는 대신 각종 장치의 인터페이스를 간단하게 가능
  - 전송 경로가 여러 개라서 전체 전송률 매우 높음
  - 장치를 추가하면 단위시간당 처리량 더욱 높일 수 있음
  - 추가 논리회로를 두어 크로스바 교환기의 신뢰도 제고
  - 전송 경로 다양하여 시스템 분할 용이

2. 다중 처리 시스템의 연결 방법

- 크로스바 교환 행렬 시스템

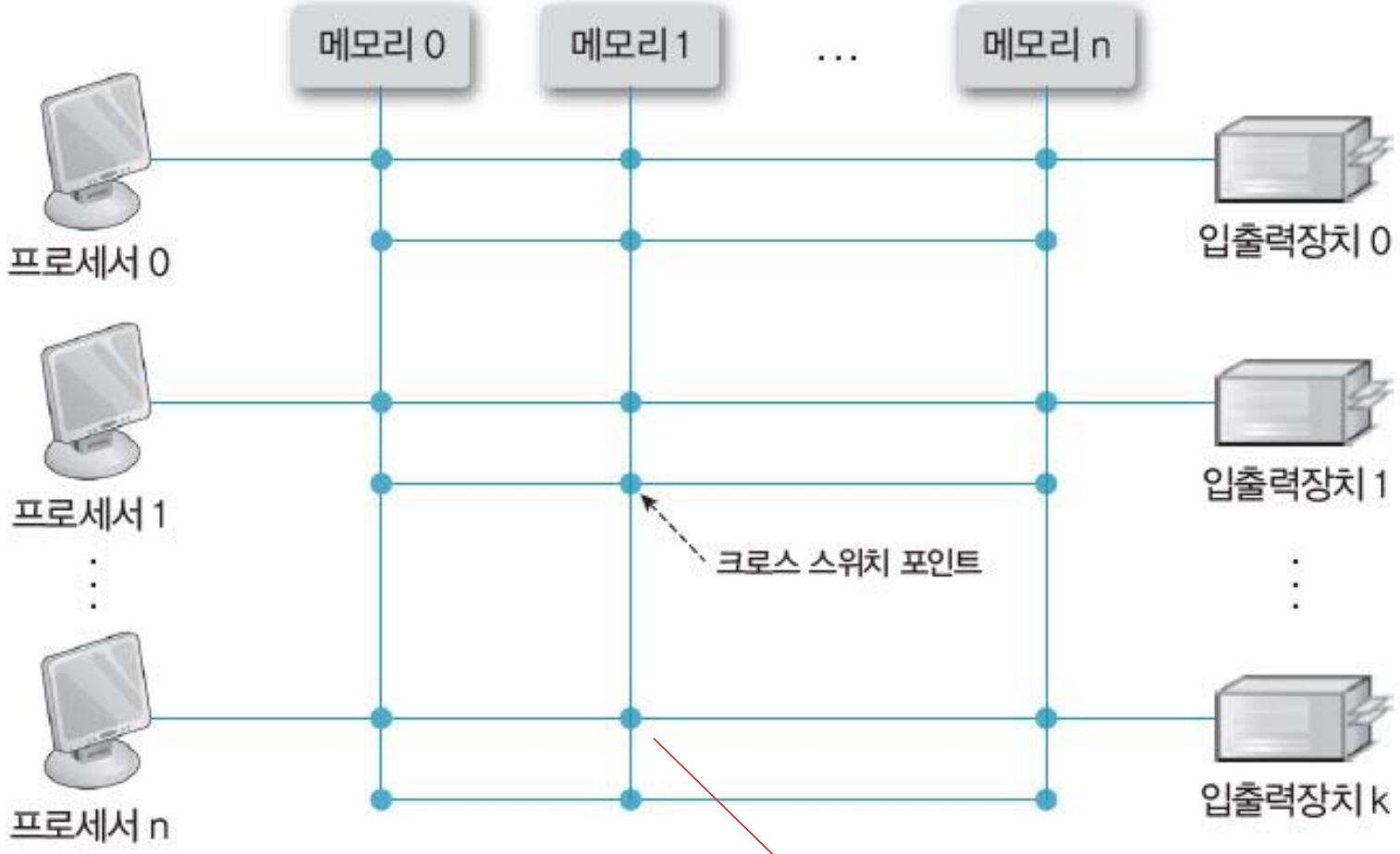


그림 11-29 **크로스바 교환 행렬 시스템**

2. 다중 처리 시스템의 연결 방법

■ 다중 포트 메모리multiport storage 시스템

- 크로스바 교환기에서 제어 논리회로와 교환 논리회로, 우선순위 조절 논리회로를 빼서 각 메모리의 인터페이스에 설치
- 각종 장치는 고유한 메모리 포트로 메모리를 참조할 수 있으며, 각종 장치마다 고유한 메모리 포트 할당
- 프로세서와 메모리 사이의 다중 경로는 높은 전송속도
- 비싼 메모리 제어 논리와 케이블, 커넥터의 수가 필요

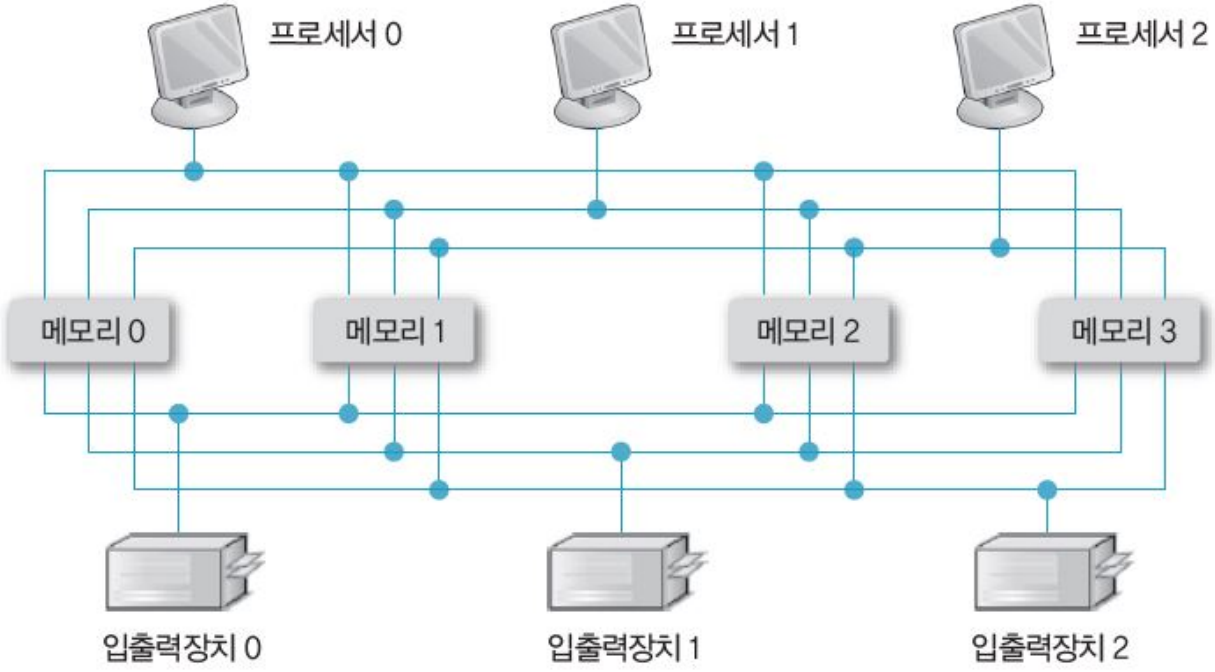


그림 11-30 다중 포트 메모리 시스템

2. 다중 처리 시스템의 연결 방법

- 메모리 포트는 보통 영구적인 우선순위 할당
  - 동시에 동일한 메모리를 참조하려는 각종 장치 간의 충돌 해결
- 프로세서가 어느 정도 메모리를 확보할 수 있으므로 매우 높은 안전성을 유지해야 하는 시스템에서 유용
- 각 메모리 참조를 몇몇 프로세서와 입출력 프로세서로 제한한 예

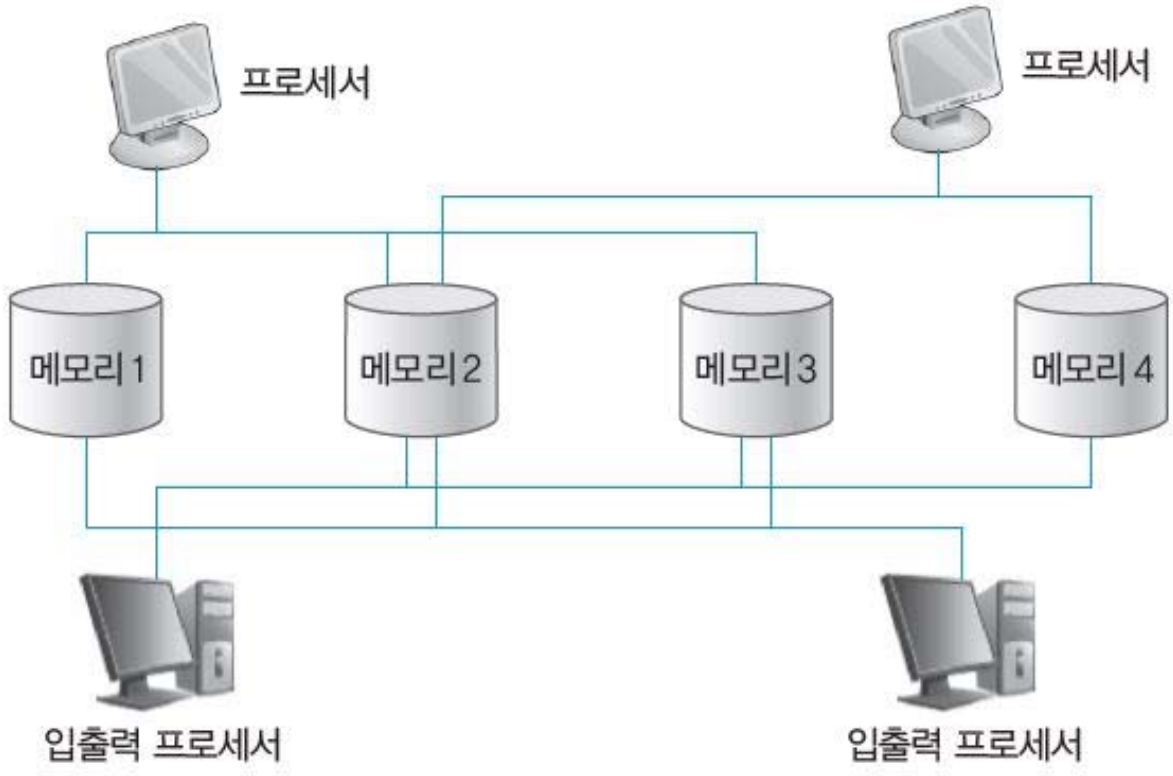


그림 11-31 **다중 포트 메모리 시스템**

2. 다중 처리 시스템의 연결 방법

하이퍼큐브hypercube 시스템

- 프로세서 N=2n개로 구성된 약결합 시스템
- 각 프로세서가 큐브의 노드를 형성. 프로세서가 필요한 각 노드를 참조하는 효과는 프로세서뿐만 아니라 지역 메모리와 입출력 인터페이스에도 포함.
- 각 프로세서에는 인접한 다른 프로세서와 직접 통신하는 경로 있음. 이 경로는 큐브의 가장자리에 대응, 프로세서에 할당 할 수 있는 이진 주소는 2n
- 하이퍼큐브 n=1, 2, 3 구조의 예

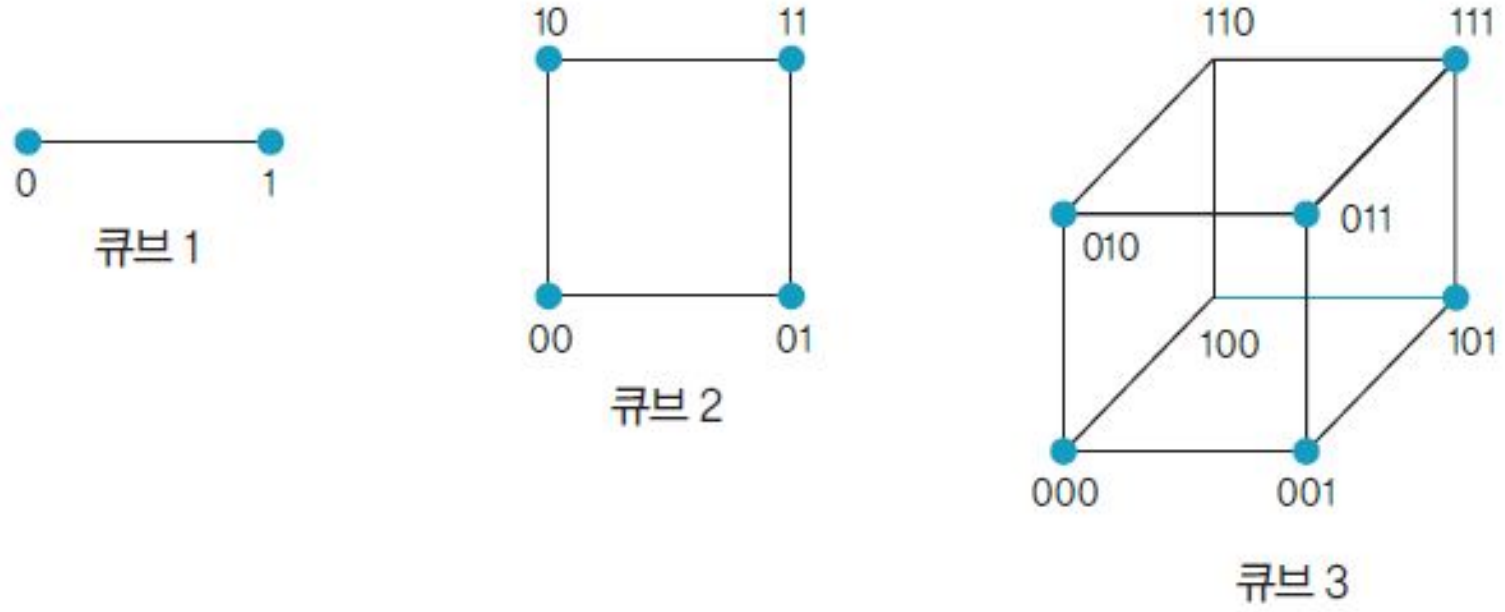


그림 11-32 하이퍼큐브 시스템 구조 예

3. 다중 처리 시스템의 운영 체제

■ 주종master/slave 운영체제

- 가장 구현하기 쉬운 구조, 현존하는 다중 프로그래밍 시스템을 간단히 고쳐서 구성 가능
- 구성

그림 11-33 주종(M/S) 다중 처리 구성

- 주(M) 프로세서는 범용 프로세서로 연산, 입출력도 담당, 종(S) 프로세서는 연산만 담당
- 주(M) 프로세서 고장 나면 전 시스템 가동할 수 없게 하므로 다른 방법보다 신뢰성 낮음
- 하드웨어의 비대칭성이 문제점. 모든 프로세서가 입출력 연산을 수행할 수 있는 대칭성이 있는 시스템과 달리 프로세서들이 동등하지 못하고 주(M) 프로세스에 종속되어 수행하므로 하드웨어 최적으로 사용 못함

3. 다중 처리 시스템의 운영 체제

■ 분리 실행

- 각 프로세서마다 운영체제가 서로 다름
- 각 프로세서에서 발생하는 인터럽트는 해당 프로세서에서 해결하는 구성 방법
- 전체 시스템에서 사용하는 정보 저장하는 테이블에 액세스하는 것은 상호배제 방법 사용 조정
- 주종 구성보다 신뢰성 높아 한 프로세서가 고장이 났다고 전 시스템 동작 못하는 것은 아니나, 고장 난 프로세서 재가동하려면 많은 작업 필요
- 구성

그림 11-34 **분리 실행 구성**

3. 다중 처리 시스템의 운영 체제

대칭

- 가장 설계하기 복잡한 구조이면서 가장 강력
- 모든 프로세서가 동등, 운영체제는 모든 프로세서, 입출력장치, 기억장치 사용하도록 관리
- 여러 프로세서가 한 운영체제 동시에 수행 가능하여 재진입 코드와 상호배제 필요. 그리고 대칭적이므로 다른 구성보다 작업 부하 효과적 분산
- 구성

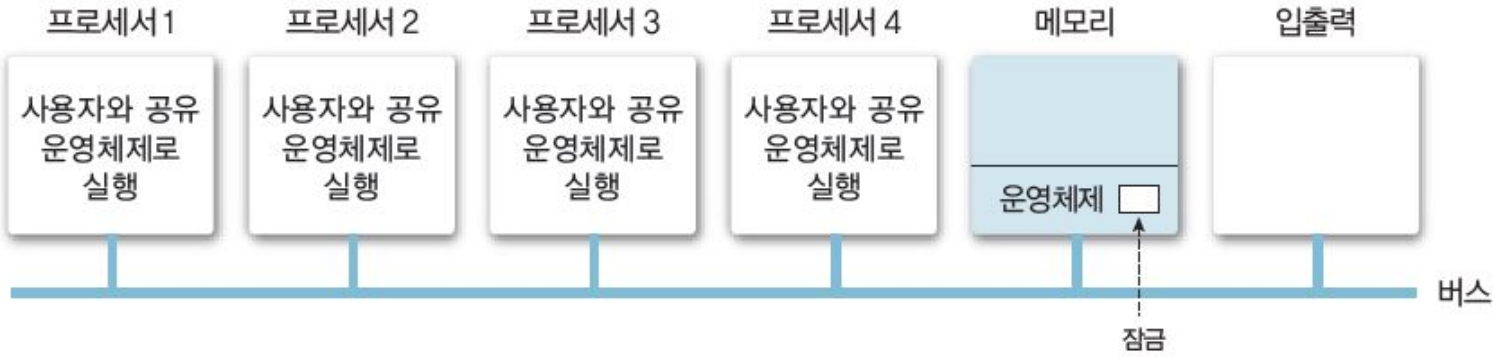


그림 11-35 **대칭 구성**

- 신뢰성이 가장높고, 한순간에 프로세서 하나만 운영 프로세서가 되게 하여 시스템의 전체 정보를 통일성 있고 일관성 있게 운영
- 자원 활용도 높음

4. 클러스터

클러스터의 구조

- 네트워크로 컴퓨터 여러 대를 연결하여 마치 단일 컴퓨터처럼 동작하도록 하는 것
- 대칭형 다중 처리를 할 수 있는 대안으로 높은 성능과 확장성 제공
- 노드 여러 개로 구성할 수 있고, 다중 프로세서도 될 수 있음
- 새로운 컴퓨터를 클러스터에 추가하면 시스템 확장 가능
- 클러스터 내의 각 노드는 독립형 컴퓨터이기 때문에 노드 하나에 장애가 발생해도 서비스를 제공할 수 있어 가용성 높음
- 최초의 클러스터 : 1994년경 NASA에서 486 PC 여러 대를 네트워크로 연결 제작 Beowulf 컴퓨터
- 백업과 사용자 관리 등 작업은 클러스터에 있는 개별적인 노드가 아니라 주로 시스템 전반에서 발생
- 네트워크 감시, 고장 탐지와 복구, 단일 시스템 관리 등 서비스를 제공하는 특정 클러스터 소프트웨어 제공해야 하며, 이를 연결하는 서버 필요

4. 클러스터

- 클러스터의 기본 구조

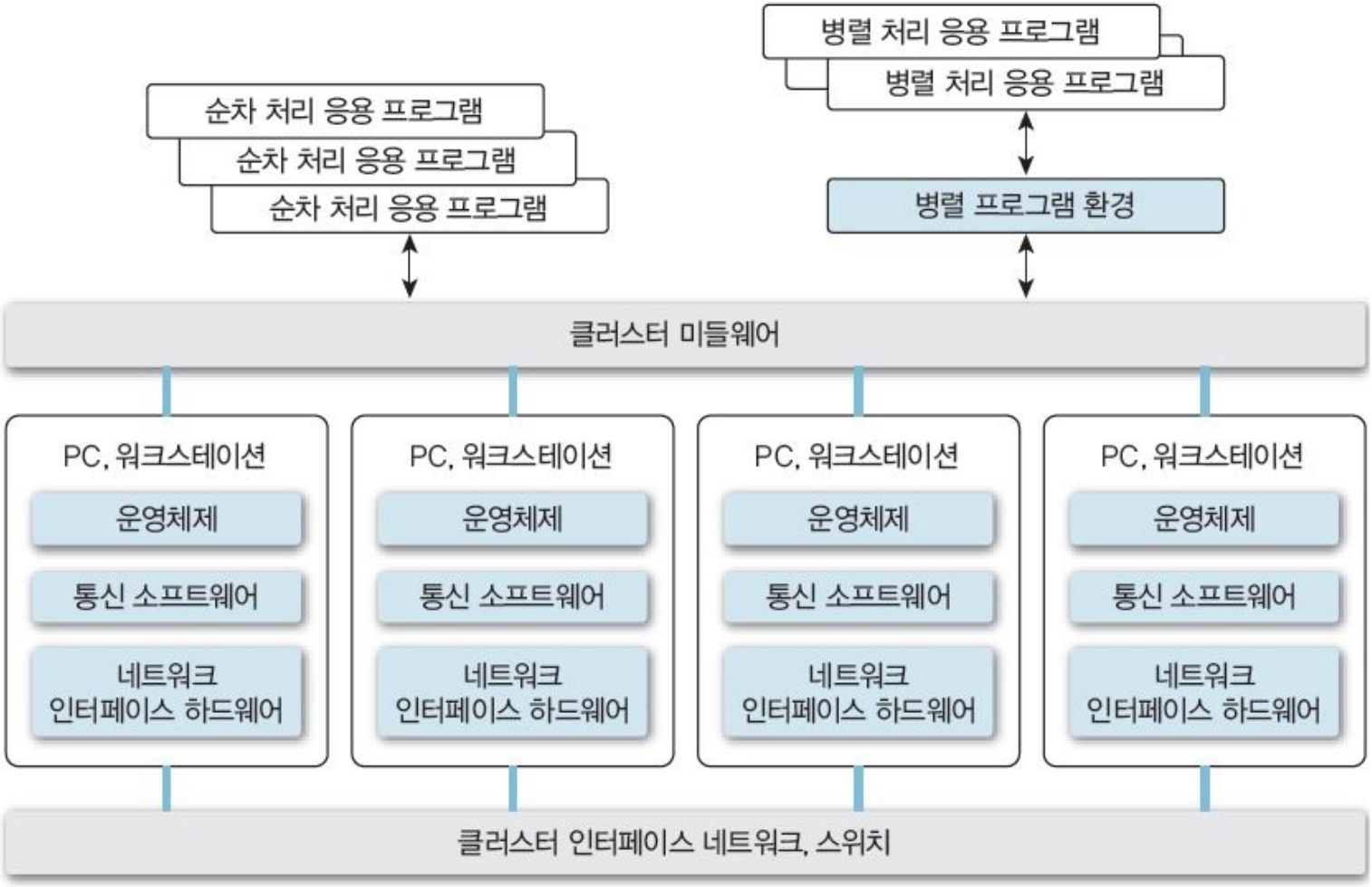


그림 11-36 **클러스터의 기본 구조**

4. 클러스터

- 클러스터에서 서버 노드는 외부 네트워크를 위해 공인 IP를 사용, 클러스터에서 통신하는 데 사용하는 IP는 모두 사설 IP
- 통신 소프트웨어는 빠르고 신뢰성 있게 통신할 수 있도록 액티브 메시지와 같은 통신 프로토콜을 사용
  - 액티브 메시지active message : 병렬 분산 시스템에서 널리 사용하는 메시지에 기반을 둔 통신 시스템
    - 핸들러를 포함하여 전송하므로, 메시지가 도착하면 핸들러가 메시지 처리. 도착한 메시지를 핸들러가 동시에 처리하기에 버퍼링으로 오버헤드 줄어듬. 또 비동기적으로 핸들러 수행하기 때문에 병렬화 장점
- 클러스터 미들웨어는 지역적으로 분리된 시스템, 즉 다수의 컴퓨터로 구성된 클러스터 시스템을 사용자가 통합된 하나의 시스템으로 인식하도록 SSISingle System Image 기능 제공하여 사용률 높임

4. 클러스터

■ 고성능 클러스터HPC, High-Performance Clusters

- 대규모 연산을 계산하는 데 사용하는 가장 일반적인 구조
- 클러스터의 노드에 다양한 연산 작업 분할, 향상된 성능 제공, 과학적 응용 프로그램 처리
- 한 노드를 계산한 중간 결과를 다른 노드에서는 미래의 계산에 적용
- 기상 예측, 핵폭발 시뮬레이션 등 수치 연산이 목적인 분야와 3D 애니메이션, 영화의 특수 효과에 사용. 대부분 과학기술 분야에서 사용, 모두 슈퍼컴퓨터 응용 분야에 해당
- 예

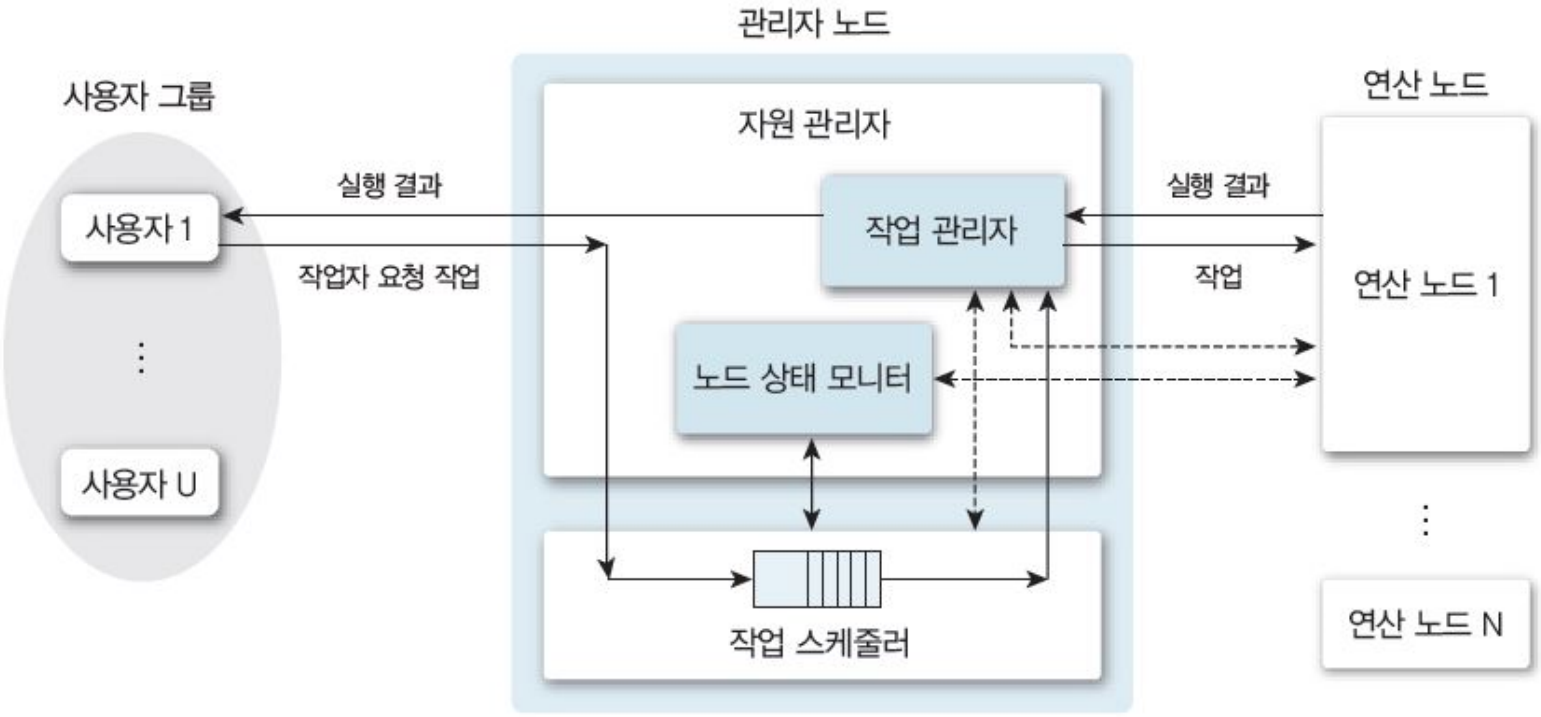


그림 11-37 클러스터의 자원 관리 시스템

4. 클러스터

- 자원 관리자
  - 기본으로 노드 상태의 모니터링과 요청 작업 수신
  - 작업 요청을 노드에서 실행할 수 있도록 할당
  - 일부 자원 관리자는 기본 스케줄링이나 정책 제어 수행
  - 복잡한 클러스터 환경에서 자원 관리자는 시스템 사용률 20~70% 상향

- 작업 스케줄러
  - 자원 관리자에게 작업을 언제, 어디서 실행할지 알려 줌
  - 우선 작업, 자원 스케줄링, 정책 및 조직의 목표 등을 통합하여 우선순위 결정
  - 스케줄러는 사용자가 작업을 제시간에 빠르고 올바르게 처리할 수 있도록 도와줌
  - 스케줄러는 사용 가능한 자원과 작업 큐, 즉 자원 관리자에게서 입력받아 순서 결정, 스케줄 확정, 작업 실행

4. 클러스터

과학 계산용 클러스터

기본형 클러스터

- 모든 노드의 하드디스크에 독립적으로 운영체제 모두 설치하여 각 노드가 시스템에 필요한 파일과 라이브러리를 자체 해결할 수 있도록 구성하는 방법

디스크 없는 클러스터

- 서버 노드 한 대에만 하드디스크가 있어 다른 노드는 서버 노드의 파일 시스템을 사용하는 diskless 방법

4. 클러스터

■ 부하분산 클러스터LBC, Load-Balancing Clusters

- 한 사용자가 요구하는 작업량은 크지 않지만, 동시에 사용자 수천 또는 수만 명이 요구하여 컴퓨터 한 대로는 처리할 수 없을 때 사용하는 구조
- 단일 서버를 사용할 때 발생할 수 있는 부하를 다른 노드로 분산시켜 웹 서버의 과부하를 해결할 수 있는 방법 제공

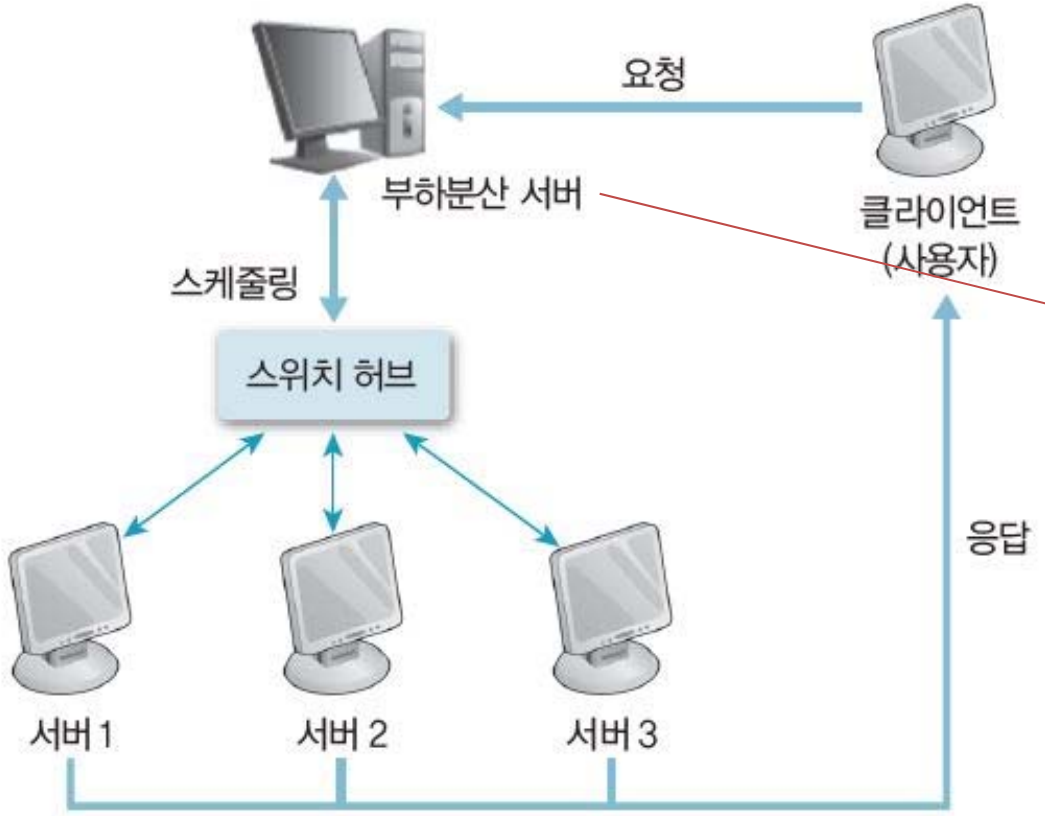


그림 11-38 부하분산 클러스터

부하 분배기 load balancer
부하분산 서버는 사용자가 요구할 때마다 최적의 부하분산을 위해 서비스를 처리하는 여러 노드 중에서 현재의 부하량이 적은 노드를 선택하여 사용자의 요청을 전달, 또 특정 노드에 결함이 발생하면 나머지 노드 적절하게 분배

4. 클러스터

■ 부하분산 클러스터의 구현 방법에 따른 구분

▪ DR(Direct Routing) 방법

- 클라이언트에 들어온 요청을 부하분산 서버가 다른 컴퓨터에 분배하고, 요청 할당받은 컴퓨터가 직접 응답하는 방법
- 클러스터의 실제 서버는 모두 공인된 IP를 사용해야 함

▪ NAT(Network Address Translation) 방법

- DR 방법과 비슷하지만, 클라이언트의 요청을 처리하는 실제 서버는 부하분산 서버를 이용하여 응답을 목적지에 보내는 것 다름. 서버에서 요청을 전달받은 실제 서버는 요청을 처리하여 다시 부하분산 서버에 응답을 보내고, 부하분산 서버는 응답을 요청한 사용자에게 전송
- 실제로는 서버가 모두 사설 IP 사용하기 때문에 공인 IP가 실제 서버 수만큼 필요하지 않고, 부하분산 서버 한 대만 공인된 IP가 필요
- 부하분산 클러스터에서 부하분산 서버가 정지하거나 고장 발생하면 부하분산 클러스터의 작동이 멈추는 문제 발생
  - 부하분산 서버를 감시하여 정지 및 고장이 발생할 때 클러스터의 다른 컴퓨터가 대신 부하 분산 서버의 역할을 할 수 있도록 하는 고가용성 클러스터를 함께 사용하여 해결

4. 클러스터

부하분산 서버의 응답을 위한 실제 서버 선택 방법

- 순환 할당 스케줄링
  - 프로세스들이 자원을 공정하게 사용할 수 있도록 각 프로세스에 일정 시간을 할당, 할당된 시간이 지나면 그 프로세스는 잠시 보류한 후 다른 프로세스에 기회를 주는 라운드 로빈 방법 사용
  - 클러스터의 구성 서버들은 서비스 처리 용량과 관계없이 동등, 각 노드에 모두 서비스 기회 제공
- 가중치 기반 순환 할당 스케줄링
  - 실제 서버에 처리 용량을 다르게 지정 가능하면 각 서버에 가중치 부여, 가중치에 따라 요청 할당
- 최소 연결 스케줄링
  - 가장 연결이 적은 서버에서 요청을 직접 연결하는 방법
  - 각 서버에서 동적으로 실제 연결한 숫자를 계산해야 하는 동적 스케줄링.
  - 연결클 때도 효과적으로 분산 수행
  - 처리 용량이 다양한 서버로 구성되어 있을 때는 효율적 부하분산 못할 수도 있음
- 가중치 기반 최소 연결 스케줄링
  - 최소 연결 스케줄링의 한 부분으로, 실제 서버에 성능 가중치를 부여하여 가중치가 높은 서버에 더 많은 요청 할당
  - 가중치의 비율인실제 연결자 수에 따라 네트워크 연결을 할당한다.

4. 클러스터

고가용성 클러스터HAC, High-Availability Clusters

- 클러스터가 제공하는 서비스의 가용성을 기본적으로 개선하려는 목적에서 구현
- 시스템이 실패하면 중복 노드 운영하여 서비스 제공
- 구조

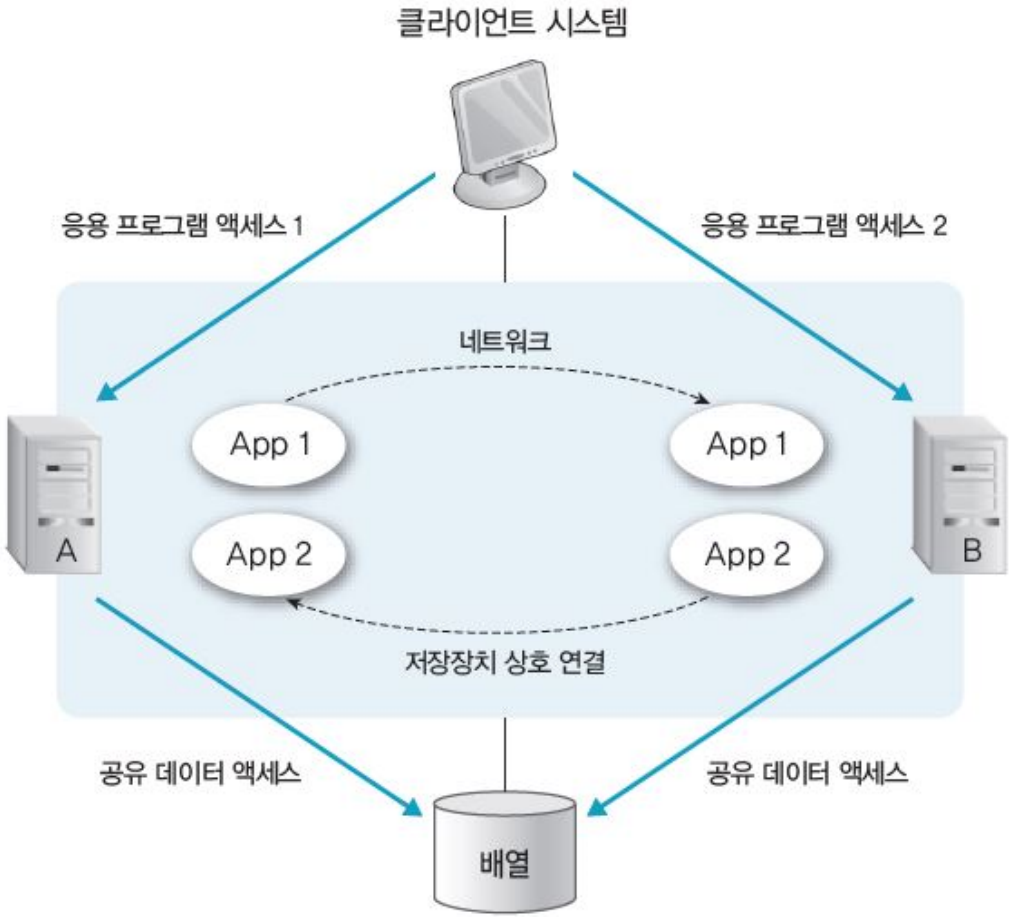


그림 11-39 **고가용성 클러스터**

4. 클러스터

클러스터의 성능에 미치는 요소

- 네트워크의 성능(전송속도와 신뢰도 포함)
  - 클러스터의 각 컴퓨터끼리는 수시로 정보를 주고받아야 하므로, 이를 얼마나 빨리 대규모로 할 수 있느냐에 따라 클러스터의 성능 다름
- 프로세서의 성능
  - 노드와 요청 작업이 많을 때 각 노드는 정보를 받으려고 대기 시간 증가(네트워크 병목 현상)
  - 가용성과 성능 향상을 위해 클러스터 시스템 구성·확장 때 병목 현상 발생하지 않도록 노드 추가하거나 디스크 배열RAID 시스템 추가
- 입출력 중심인 응용 프로그램은 클러스터의 어느 노드에서든 자신이나 원격 노드에 위치한 주변기기, 메모리(디스크)를 직접 액세스할 수 있도록 전역 입출력 주소 공간 지원하는 단일 입출력 공간SIOS, Single I/O Space 구현해야 함
- 여러 노드를 상호 연결하여 구성하는 클러스터 시스템을 통합된 자원으로 사용할 수 있도록 SSI 기능을 제공하여 편의성, 확장성, 신뢰성 측면에서 클러스터 이용률 높임