제8장 성능 향상을 위한 인프라 구조

8.3 3계층형 시스템 그림을 통해 본 병목 현상

8.3.3 디스크 I/O 병목 현상
8.3.4 네트워크 I/O 병목 현상
8.3.5 애플리케이션 병목 현상 예(ex)

8.3.3 디스크 I/O 병목 현상

- I/O 병목 현상은 하드 디스크 등의 저장 장치에 대한 I/O 병목이다.

  I/O가 병목 지점이 될 때는 CPU 수를 늘리거나 클럭 주파수를 높여도 효과가 없다.

  I/O 효율을 높이든가 I/O를 줄이는 방법을 고민해야 한다

외부저장소

-기업에서는 데이터베이스의 저장 위치로 외부 저장소를 사용한다. Storage Area Network(SAN) / 네트워크를 경유하는 Network Attached Storage(NAS) 저장소가 있다

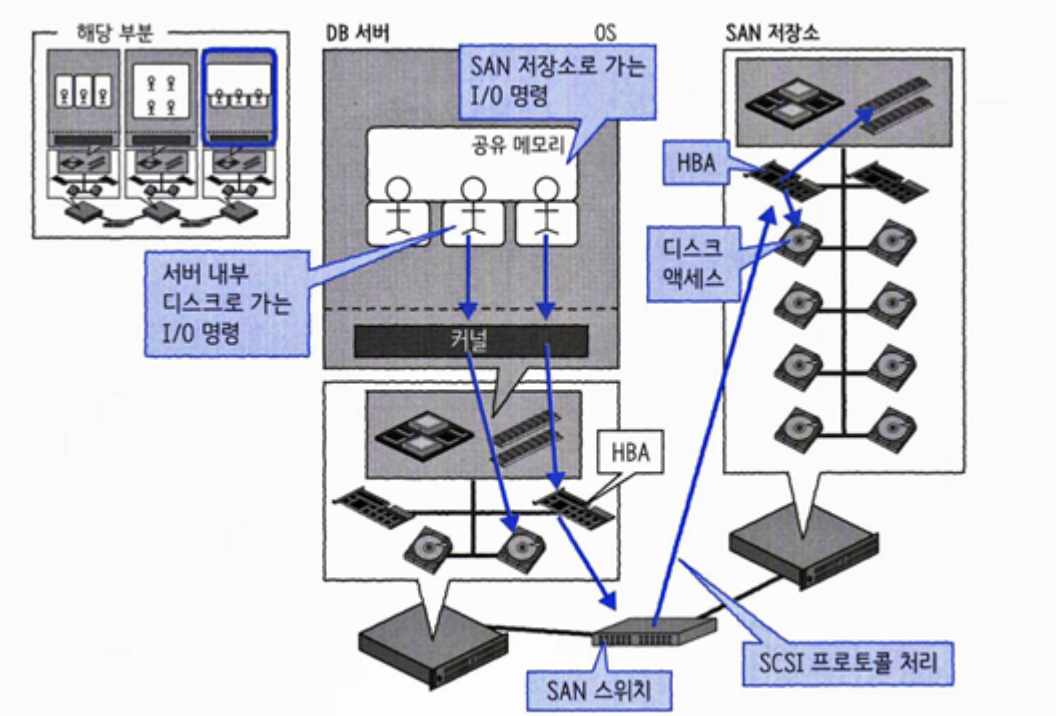

그림 8.21 로컬 디스크와 SAN 저장소

이 그림은 DB 서버가 SAN 저장소에 접속하는 것을 나타낸다.

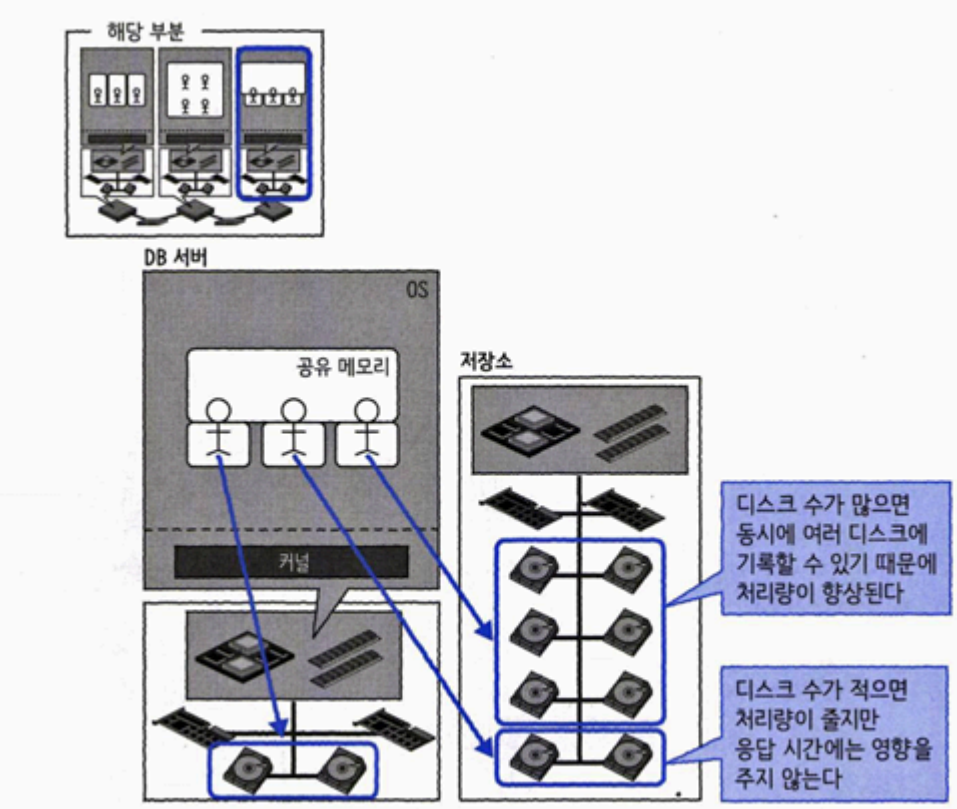

그림 8.22 RAID를 구성하는 디스크 수가 많을수록 처리량이 높아진다

성능이라는 관점에서는 대부분의 로컬 디스크는 3~4대의 디스크로 RAID(**여러 개의 물리적 하드 드라이브를 논리적으로 묶어 저장 장치의 성능**)를 구성하고, 캐시로는 서버의 OS 메모리를 이용한다. 이에 비해 외부 저장소는 수십 대에서 수백 대 단위의 디스크를 배치하고, 거기에 캐시 전용 메모리 영역까지 갖추고 있다. 디스크 수에 따라 처리량이 증가하므로 처리량 관점에서는 외부 저장소가 압도적으로 유리하다

처리량은 동일 영역을 이용하고 있는 사용자가 많을수록 저하된다. 하나의 RAID 그룹을 독점할 수 있으면 좋지만, 다른 애플리케이션과 영역을 공유하고 있는 경우도 있다.
이것은 OS 및 저장소 설정에 의존하므로 애플리케이션 측면에서 조정이 어려워 주의가 필요하다.

순차 I/O와 랜덤 I/O

- 디스크 I/O에는 순차 (Sequential) 액세스와 랜덤(Random) 액세스가 있다.

순차는 순서를 따른다는 의미로, 선두부터 차례대로 액세스(읽기/쓰기)하 는 방식이다.

랜덤 액세스는 헤드가 움직이면서 해당 위치로 바로 건너뛰는 액세스(읽기/쓰기) 방식이다.

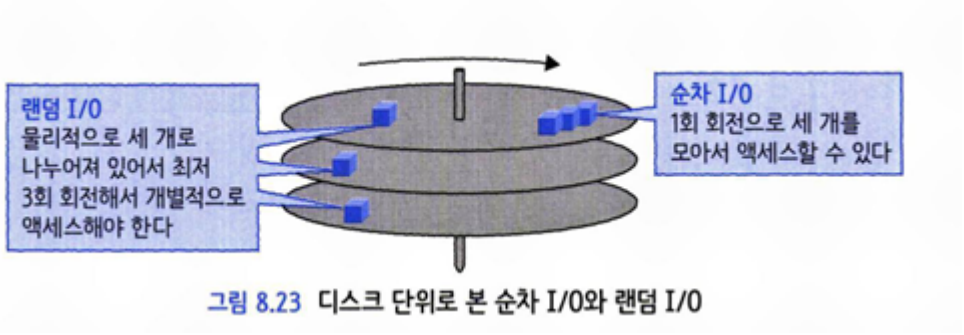

그림 8.23 디스크 단위로 본 순차 I/O와 랜덤 I/O

그림 8.23과 같이 CD나 DVD를 예로 들면, 순차 액세스는 최고속으로 빠르게 회전하고 있는 형태이고,

랜덤 액세스는 항상 해당 부분을 찾고 있는 형태다. 해당 위치를 찾는 것은 대상 위치를 찾아서 바늘을 움직여야 하기 때문에 시간이 걸린다

단일 디스크가 기록 위치인 경우는 순차 방식이 빠르고 랜덤은 느리다.

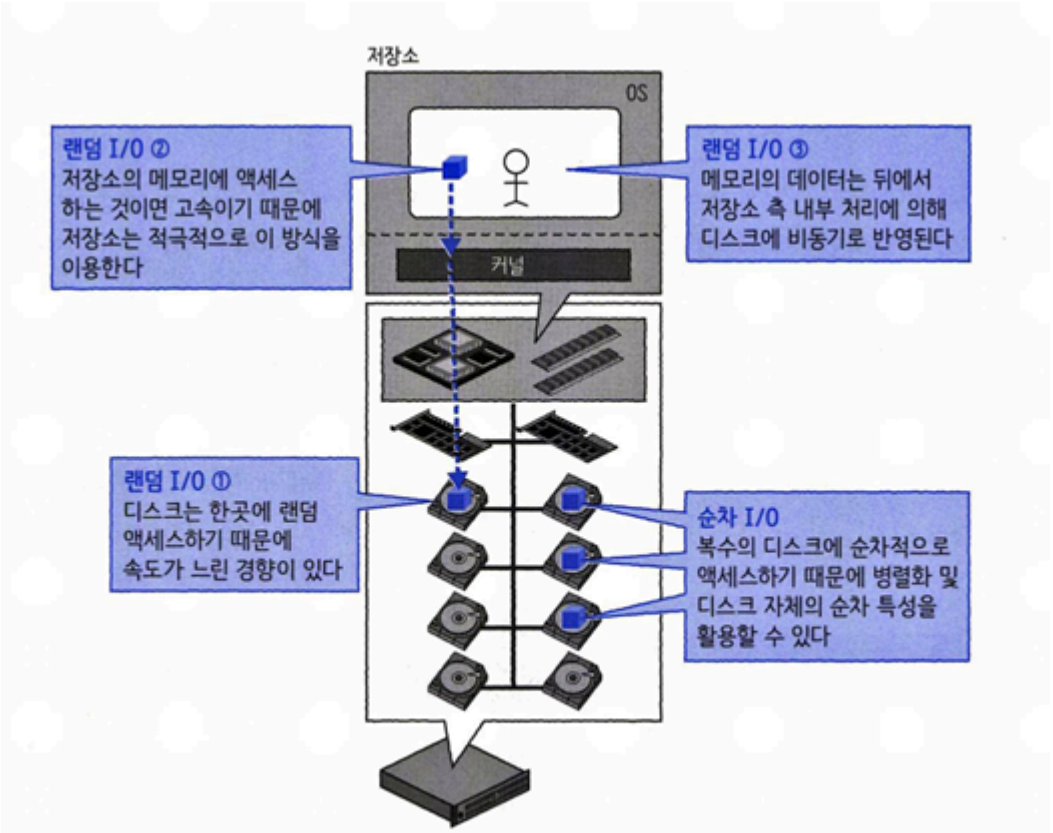

그림 8.24 저장소 계층에서 본 순차 I/O와 랜덤 I/O

그림 8.24.는 기업 시스템에서 이용하는 저장소는 내부에 다수의 디스크를 가지고 있는데 랜덤과 순차를 활용한 그림이다

저장 장치 측에서는 큰 파일에 대한 일괄 읽기 처리가 발생하면 다수의 디스크에 대해 동시에 순차 액세스를 한다. 이를 통해 순차 및 병렬로 액세스할 수 있어서 대용량 데이터라도 고속으로 처리할 수 있다

하지만 작은 파일에 액세스하는 경우에는 단일 디스크에 랜덤 I/O를 하기 때문에 그다지 빠르지 않다. 때문에 저장 장치 측에서는 메모리나 SSD 등 랜덤 I/O에 강한 기억 영역을 이용하여 디스크의 내용을 캐시하는 구조를 적용해서 효율화를 도모하고 있다.

순차/랜덤 특성은 파일 크기에 의존하지 않는다.

예를 들어, 같은 크기의 데이터에 액세스하는 경우라도 데이터가 여기저기 분산돼서 몇 번이고 위치를 찾아야 한다면 느려진다

8.3.4 네트워크 I/O 병목 현상

- 네트워크를 경유한 I/O는 CPU 버스나 메모리 간 I/O보다도 응답 시간 오버헤드가 크다. 이 때문에 응답을 근본적으로 개선하는 것은 어려우며, 처리량을 개선하는 접근법 이나 네트워크 I/O 자체가 발생하지 않도록 하는 방법이 효과가 있다.

통신 프로세스의 병목 현상

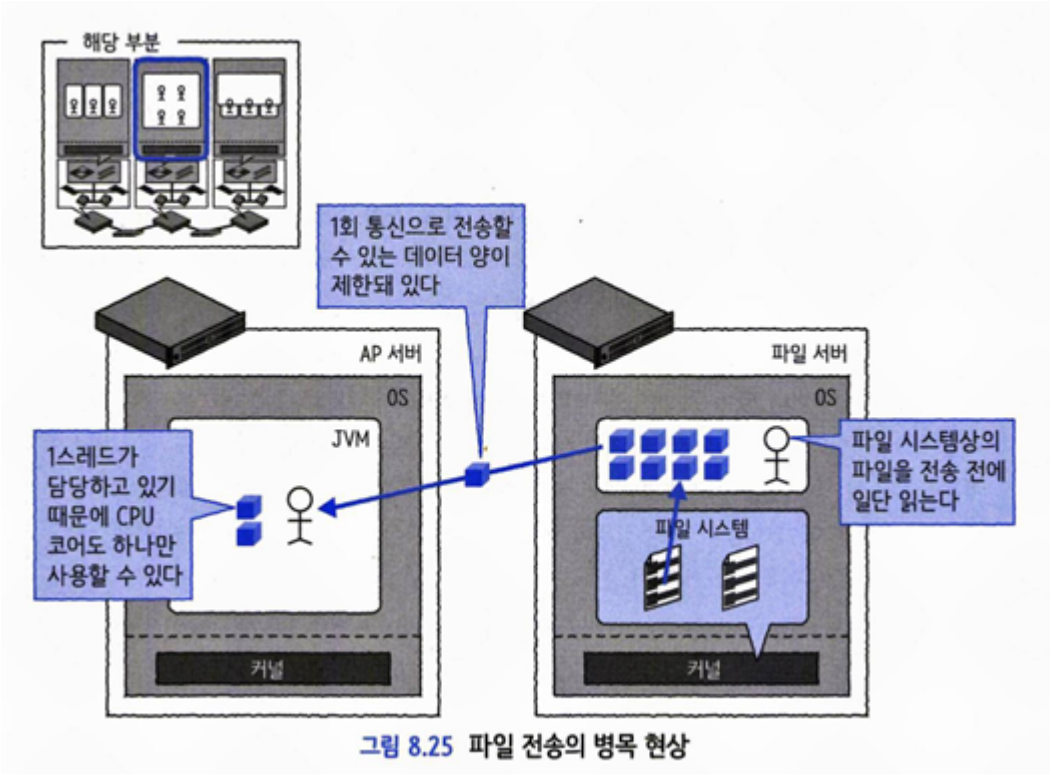

그림 8.25 파일 전송의 병목 현상

그림 8.25처럼 하나의 프로세스로 처리하는 경우 높은 처리량을 실현하는 것이 매우 어렵다. 이유는 통신에는 반드시 '데이터 전송', '통신 결과 확인' 같은 처리가 포함되므로 항상 풀 파워로 송수신이 이루어지지 않기 때문이다. 또한, 통신이 고속화되면 CPU에서 병목 현상이 발생할 수도 있다

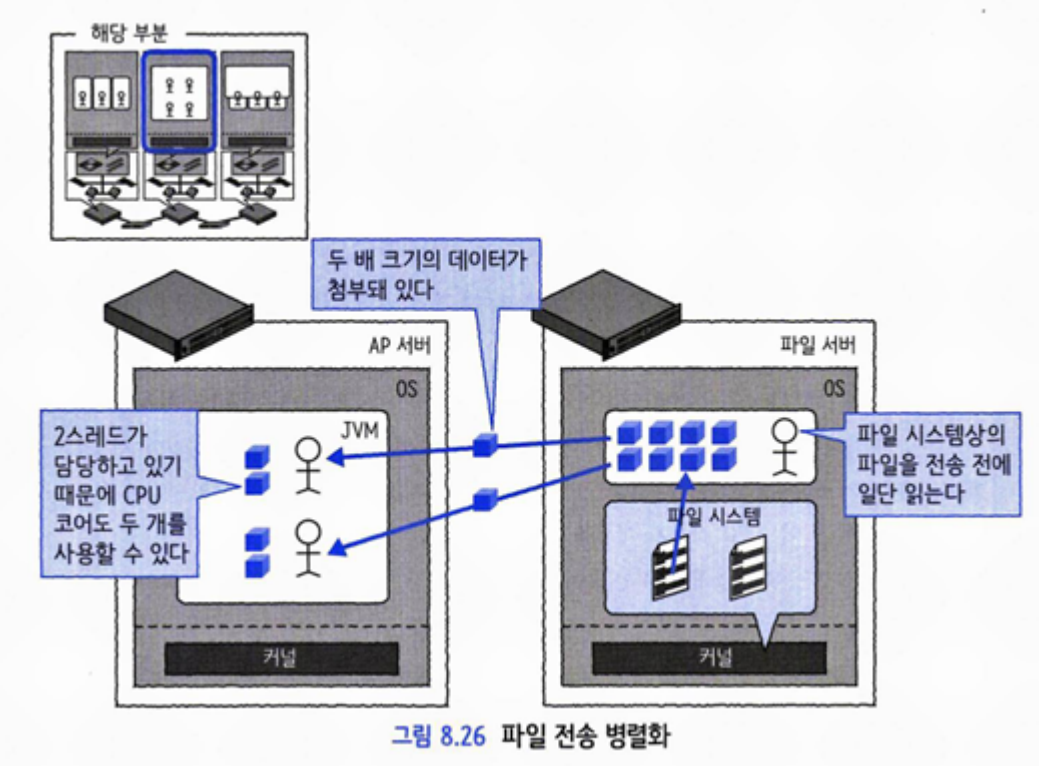

그림 8.26 파일 전송 병렬화

통신에서 대역을 모두 사용하려면 그림 8.26과 같이 처리를 다중화해서 병렬화를 해야 한다. 다중화 할 수록 통신량이 많아지므로 대역폭이 최대치에 가까운 처리량을 실현할 수 있다. OS나 소프트웨어에 따라서는 이 다중화를 자동으로 구현하고 있는 것도 있다.
병렬화라는 개념은 대역을 최대로 사용한다는 관점에서 매우 유용한 접근법이다. 특히, CPU의 멀티 코어화가 진행되고 있어서 처리 병렬화의 유용성이 높아지고 있다.

네트워크경로의 병목 현상

-네트워크경로의 병목 현상의 예를 하나 들어보겠습니다

ex)현재 분석 시스템이 오래돼서 최신 분석 시스템으로 교체하기로 했다

분석 시스템이기때문에 데이터베이스 I/O가 높을 것이고, 1회 보고서 출력을 위해 무거운 SQL이 실행
될 것이 틀림없다. 따라서 디스크에 가장 많은 비용을 투입했고, 분석 시스템의 AP 서버는 병렬로 확장할 수 있도록 구성했다

성능 테스트도 완료했고 최종적으로 시스템을 배포했지만, 예상한 것보다 응답 속도가 느렸다.

부하를 보니 DB 서버에는 여유가 있고 AP 서버가 비교적 리소스를 많이 사용하고 있었다. AP 서버에 병목 현상이 발생하고 있는 것이 틀림없다고 생각하고 AP 서버를 증설했지만, 성능에는 전혀 변화가 없다. 도대체 무엇이 문제일까?

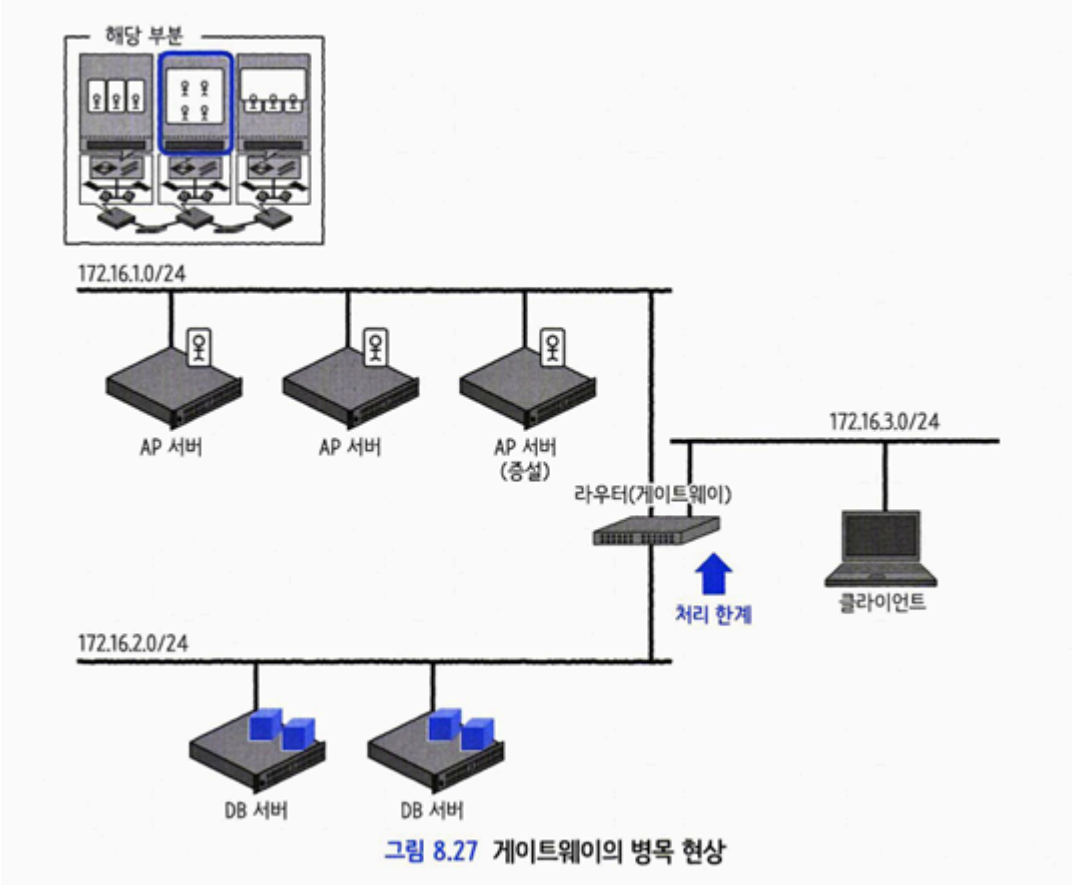

그림 8.27 게이트웨이의 병목 현상

그림 8.27에 있는 것처럼 원인은 기본 게이트웨이였다. AP 서버에서 클라이언트 PC로, DB 서버에서 AP 서버로 가는 큰 트랜잭션들이 모두 게이트웨이인 특정 라우터를 경유하는 바람에 라우터가 처리 한계에 다다랐던 것이다. 또한, 사내 여러 시스템과 클라이언트 사이의 게이트웨이 역할을 하고 있었던 것도 원인이었다.

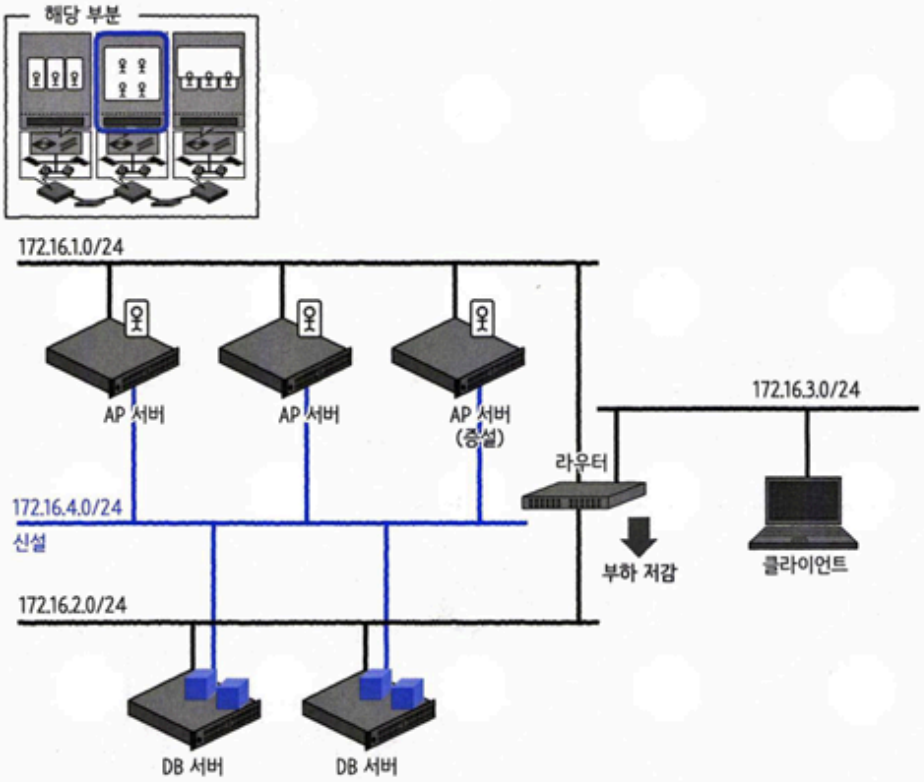

그림 8.28 게이트웨이의 병목 현상 개선

그림 8.28과 같이 AP 서버와 DB 서버용으로 전용 네트워크를 증설해서 트래픽을 분할했다. 그 결과, AP 서버에서 클라이언트로 가는 통신 분량만큼 응답 문제가 개선됐다. 라우터 중에는 방화벽 기능을 갖추고 있는 것이나, 세션 감시를 해서 장시간 지연되고 있는 세션을 강제적으로 끊어 버리는 기능 등 고기능을 갖춘 것도 있다. 시스템을 신규 구축할 때는 IP 주소 수가 부족한지만 확인하는 것이 아니라 경로와 트래픽 증감에 대
해서도 검토하도록 하자

8.3.5 애플리케이션 병목 현상 예(Ex)

데이터 갱신의 병목 현상

-데이터베이스를 이용한 시스템에서 자주 발생하는 것이 특정 데이터에 의존하는 처리가 병목 지점이 되는 것이다.

예를 들어, 판매 개수를 기록해야 하는데 '반드시 재고 개수에서 1을 뺀다'라는 처리가 있다면, 일반적으로는 테이블의 특정 레코드 값을 변경하도록 구현된다.

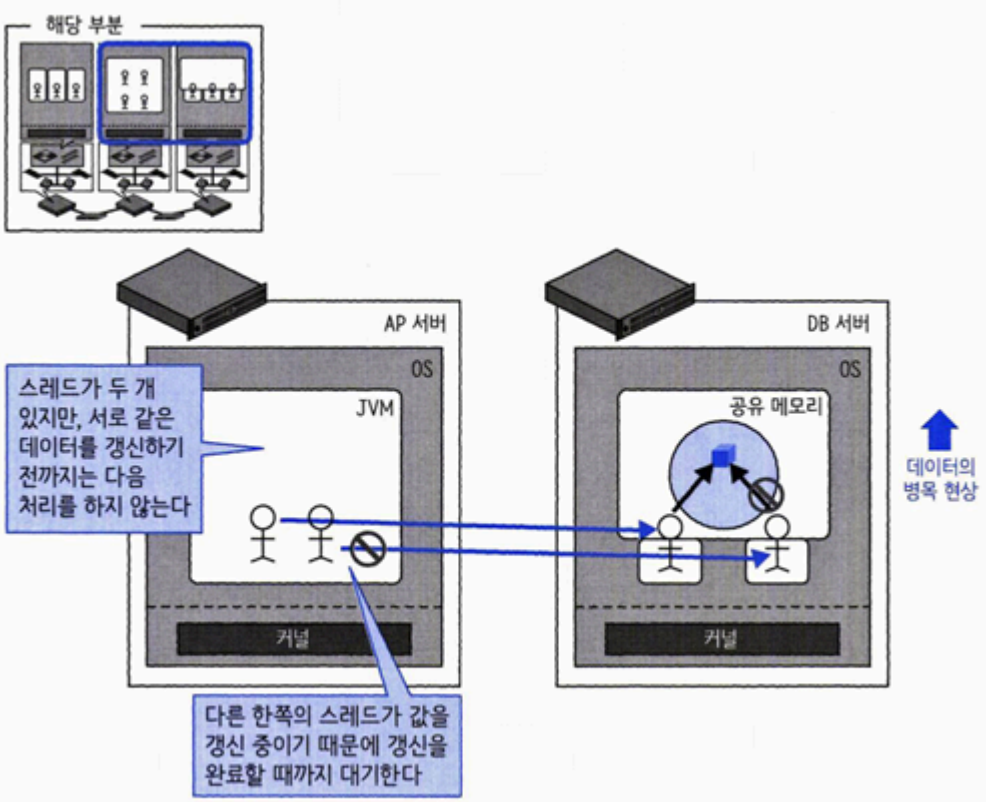

그림 8.29 모든 처리가 같은 테이블의 같은 레코드를 갱신하기 전까지는 다음 처리를 진행하지 않는다

- 값의 캐시화

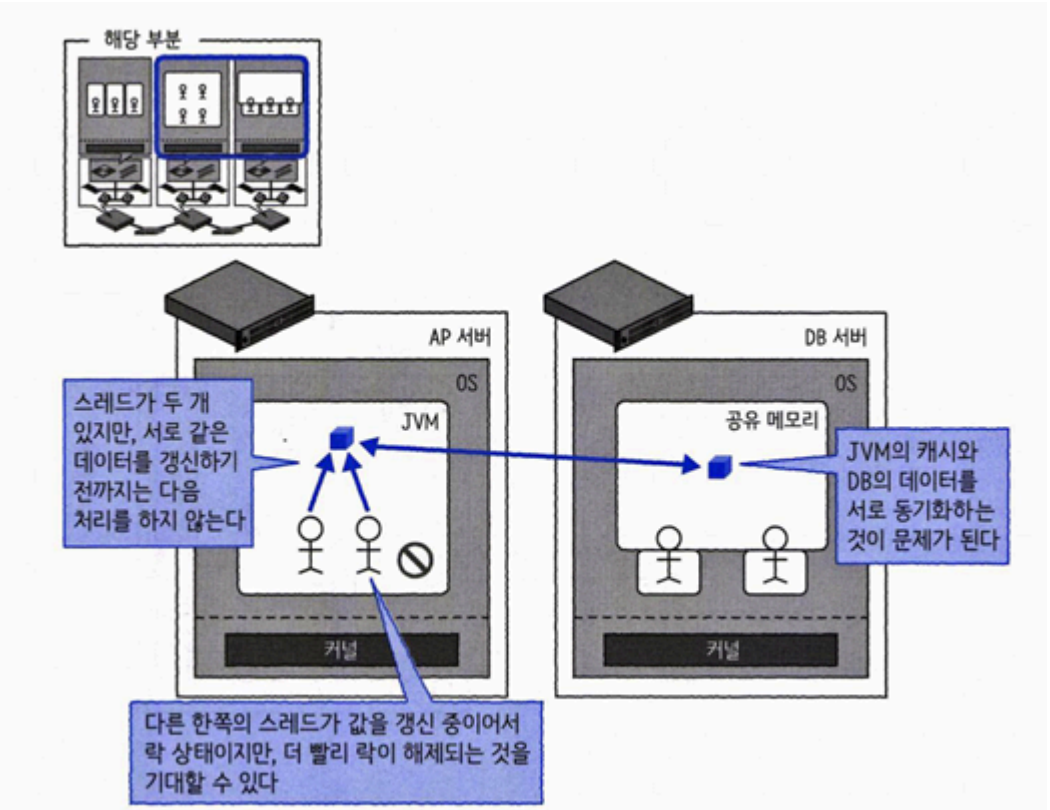

그림 8.30 값을 캐시화한다

별도 서버에 질의를 던지는 것이 병목 지점이 된다면, 더 가까운 장소에 캐시화 하는 것이 일반적인 방법이다. 단, 네트워크를 경유하는 질의가 없어지므로 처리 효율이 개선될 수 있지만, 병목 지점이라는 것에는 변함이 없어서 근본적인 해결책은 되지 못한다

- 병목 지점의 분할

재고가 200개가 있다고 하면 100개씩 레코드를 분할한다. 이 경우 한번에 두 개의 처리를 진행할 수 있어서 처리 다중화가 가능하다. 결과적으로는 재고가 몇 개 있는지 확인하는 데이터 일치성 문제와 한쪽이 먼저 고갈된 경우 데이터 최신성 문제가 새롭게 발생한다.

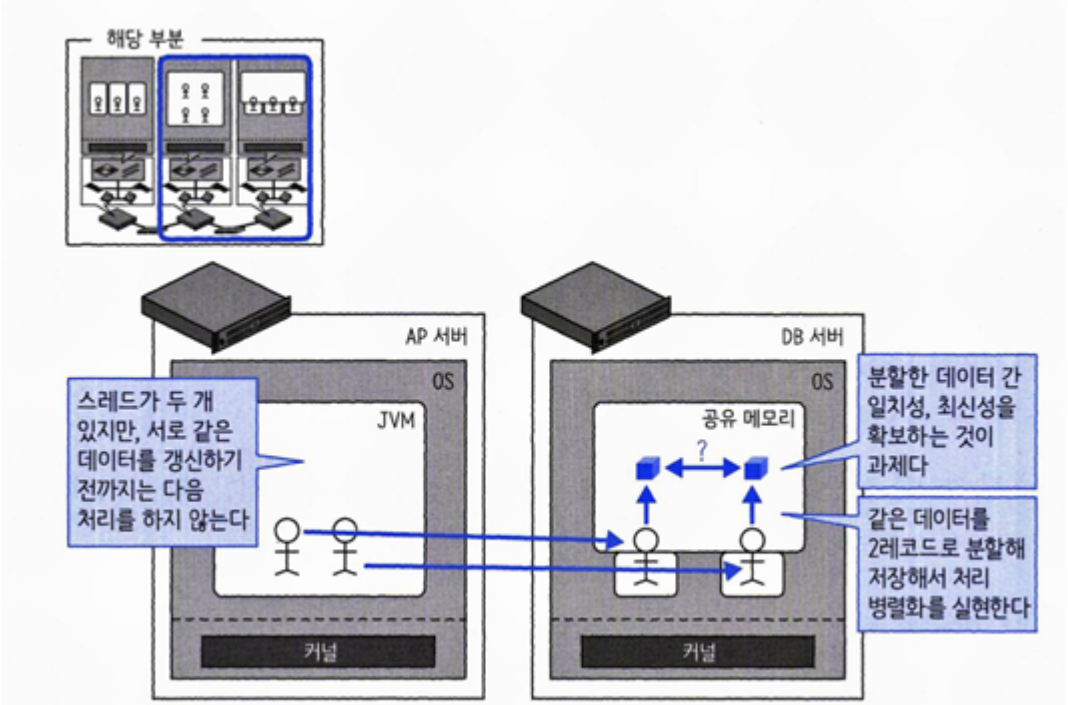

그림 8.31 병목 지점을 분할하자

참고로, 오라클 DB에서는 레코드를 분할해도 같은 데이터베이스 블록 내부에 저장될 가능성이 있으므로 병목 현상이 해결되지 않을 수도 있다. 이런 경우에는 테이블 자체를 나누든가, 메모리 병목 현상 예에서 설명한 것처럼 파티션 기능을 이용해서 각각의 레코드를 다른 파티션에 저장하는 접근법이 유용하다.

외부 질의의 병목 현상

-시스템 하나로 완성되는 시스템은 거의 없다. 대부분의 시스템은 다른 시스템과 데이터 연계 등을 통해 협력할 필요가 있다. 이 부분이 병목 지점이 되는 경우도 많다.

예를 들어, 기반 시스템의 사용자 관리를 일원화 하려는 프로젝트가 있었다. 사용자 정보는 LDAP와 Active Directory를 연계했고, 윈도우 및 리눅스/유닉스의 사용자 계정을 모두 통합해서 기존 시스템과 교체했다. 그런데 교체 후 업무용 일괄 처리에서 큰 폭의 처리 지연이 발생해서 처리가 예정 시간 내에 끝나지 않게 됐다. 원인은 무엇일까?
이 업무의 일괄 처리는 1트랜잭션을 실행 시마다 사용자 정보를 확인하는 처리였다. 그 전까지는 로컬 시스템 내에 사용자 정보가 저장돼 있어서 질의 시 시간이 걸리지 않았지만, 그림 8.32와 같이 매번 LDAP를 통해 인증을 하므로 이 처리에 병목 현상이 발생하고 있었다.

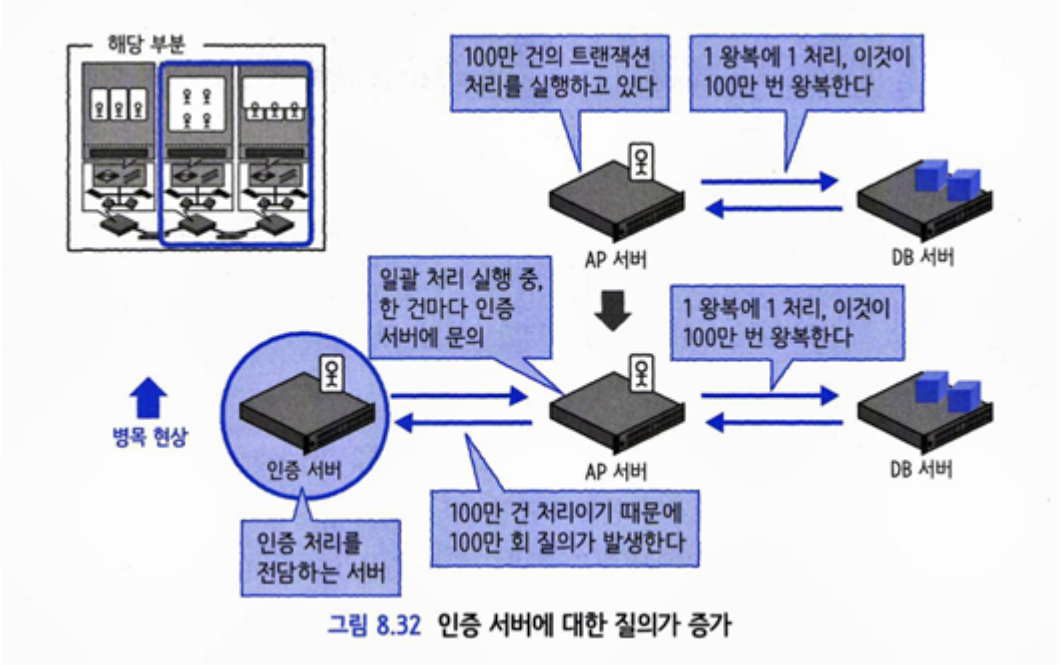

그림 8.32 인증 서버에 대한 질의가 증가

1회 질의로 1 일괄 처리 내의 사용자 정보를 취득하고, 애플리케이션 측에서 일차 파일을 캐시로 저장해서 차이만 확인하는 방법으로 변경했더니 이 현상은 사라졌다고 한다.