

chap06. 캐시

7.0 들어가며

웹 캐시는 자주 사용되는 웹 문서의 사본을 저장하여, 동일한 요청이 들어올 때 원 서버 대신 캐시된 사본으로 응답하는 HTTP 장치이다.

- 주요 장점

1. 불필요한 데이터 전송 감소로 네트워크 비용 절감
2. 네트워크 병목 감소로 페이지 로딩 속도 향상
3. 원 서버 요청 감소로 서버 부하 완화 및 응답 속도 개선
4. 원거리 서버 접근 시 지연 시간 단축

- 이 장의 주요 학습 내용

1. 캐시를 통해 성능 향상 및 비용 절감하는 방법
2. 캐시 효과를 측정하는 방법
3. 최적의 캐시 위치 선정 전략
4. HTTP가 캐시의 신선도를 유지하는 방식
5. 캐시와 다른 캐시 또는 서버 간 상호작용 방식

7.1 불필요한 데이터 전송

- 여러 클라이언트가 자주 접근하는 문서를 서버가 반복해서 전송하면, 같은 바이트가 네트워크를 통해 계속 이동하게 되어 불필요한 데이터 전송과 대역폭 낭비가 발생한다.
- 캐시는 이러한 중복 전송을 줄이고, 서버 부하를 완화하며 응답 속도를 개선한다.
- 캐시된 사본을 활용하면 원 서버로의 요청을 줄여 네트워크 병목과 거리 지연도 완화할 수 있다.

7.2 대역폭 병목

- 웹 캐시는 대역폭 병목을 줄이는 데 효과적이다.
- 로컬 네트워크는 원격 서버보다 대역폭이 크기 때문에, 캐시가 있는 경우 데이터를 훨씬 빠르게 전달할 수 있다.
- 클라이언트가 서버에 접근할 때의 속도는 사용하는 네트워크 종류에 따라 극적으로 달라진다.
 - 예시 설명
 - 샌프란시스코 지사의 사용자가 애틀랜타 본사 서버에서 5MB 문서를 다운로드할 경우, WAN을 통해 30초가 걸릴 수 있지만, 로컬 캐시에서는 1초 이내에 받을 수 있다.

네트워크 종류에 따라 전송 시간에 큰 차이가 발생하며, 대역폭이 빠른 환경일수록 캐시의 이점이 더욱 극대화 된다.

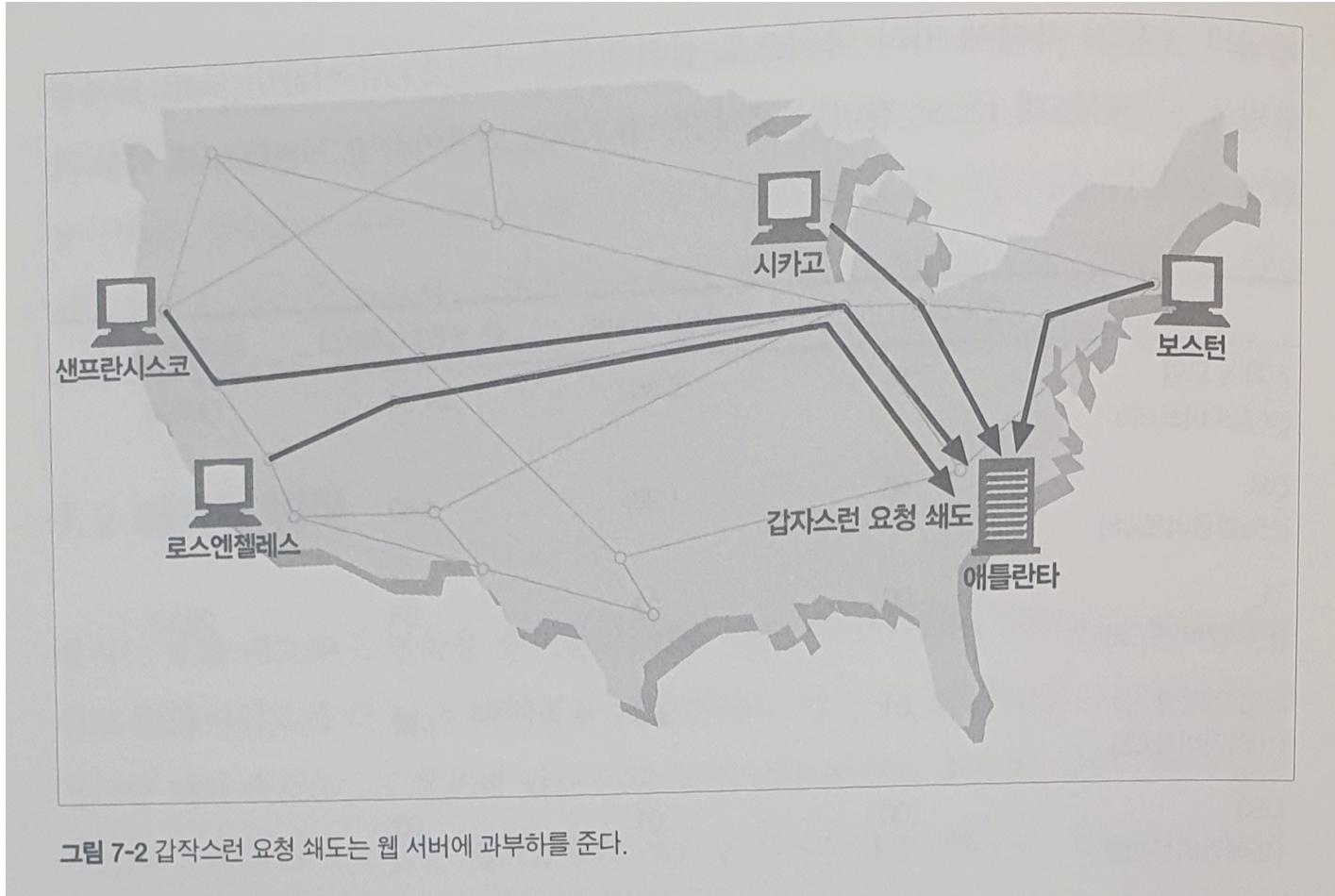
특히 대용량 문서일수록 캐시를 통한 로컬 전송의 성능 차이는 더욱 두드러진다.

7.3 갑작스런 요청 쇄도 (Flash Crowds)

- 웹 캐시는 뉴스 속보, 유명 인사 관련 사건, 스팸 메일 등으로 인해 대량의 사용자가 거의 동시에 특정 웹 문서에 접근하는 현상인 '요청 쇄도'에 대응하는 데 매우 중요하다.
- 이와 같은 트래픽 급증은 네트워크와 웹 서버에 과부하를 일으켜 장애를 유발할 수 있다.

사례 설명

- 1998년 9월 11일, 클린턴 대통령 관련 보고서("스타 보고서")가 공개되자 미 하원 웹 서버는 한 시간 동안 3백만 건이 넘는 요청을 받음. 이는 평소 트래픽의 50배에 해당.
- CNN.com은 같은 시기 웹 서버가 매초 50,000건이 넘는 요청을 수신했다고 보고함.



캐시를 적절히 활용하면 이러한 요청 쇄도 상황에서 서버와 네트워크의 부담을 분산시켜 장애를 예방할 수 있다.

7.4 거리로 인한 지연

- 대역폭이 충분하더라도, 클라이언트와 서버 간 물리적 **거리는 지연(latency)**의 원인이 될 수 있다.
- 인터넷 트래픽은 광섬유 등의 물리 매체를 따라 이동하고, **빛의 속도 제한(약 300,000km/s)** 때문에 거리 차이에 따라 전달 지연이 발생 한다.

예시>

- **보스턴 ↔ 샌프란시스코 거리:** 약 4,400km
- 이상적인 조건에서는 왕복에 약 **30밀리초**가 걸린다.
- 웹 페이지가 **20개의 작은 이미지를** 포함하고 있고, 클라이언트가 **4개의 병렬 커넥션**을 유지한다고 가정할 경우, 전체 다운로드에 **240 밀리초** 이상의 지연이 발생할 수 있다.
- 만약 서버가 더 멀리 있는 도쿄(10,800km)에 있다면, **600밀리초** 이상으로 지연은 더 커진다.
- 실제로는 웹 페이지가 복잡할수록 **수 초 단위**의 지연도 발생 가능하다.

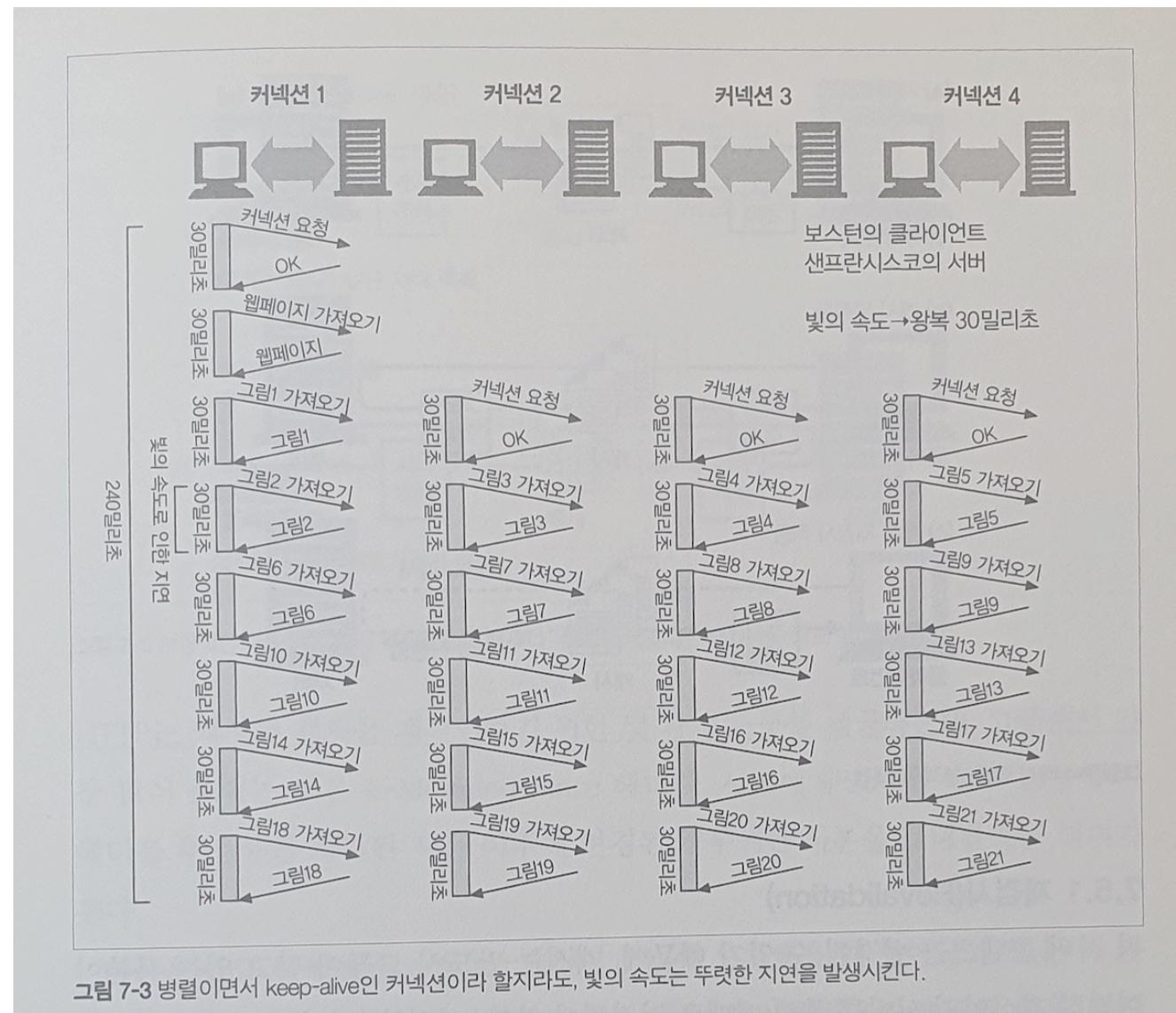


그림 7-3 설명

- 클라이언트는 동시에 4개의 연결을 통해 서버에서 리소스를 요청하지만, 각 연결마다 왕복 지연(RTT)이 반복되며 누적된다. (**RTT (Round-Trip Time, 왕복 지연 시간)**)
 - Keep-Alive 커넥션을 사용하더라도 지연은 근본적으로 거리의 영향을 받는다.

▼ keep-Alive (http1.1)

기본 HTTP/1.0 동작 방식

- HTTP 요청/응답이 끝나면 **TCP 연결을 즉시 끊습니다.**
 - 브라우저가 같은 서버에 여러 리소스(HTML, 이미지, CSS 등)를 요청할 경우, **매번 새로운 TCP 연결을 열고 닫아야** 해서 지연이 증가하고, 서버에도 부담이 큽니다.

HTTP/1.1 또는 Connection: keep-alive

- Keep-Alive는 하나의 TCP 연결을 유지하면서 여러 HTTP 요청을 처리할 수 있도록 합니다.
 - 즉, 연결을 재사용함으로써 다음과 같은 효과가 있습니다:
 - TCP 핸드셰이크(3-way handshake) 비용 절감
 - 지연 시간 감소
 - 서버와 클라이언트 모두의 리소스 절약

▼ 요즘은...? (http2, http3) - (너무 자세해 지는거 같아서 여기까지....)

http2(대부분 사용) : 하나의 TCP연결에서 멀티플렉싱 처리되며, keep-alive(연결지속)은 내장되어있다고 보면된다.

http3(선택증가) : QUIC 기반 (UDP). 연결 지연 최소화. 헤드오브라인블로킹 해소

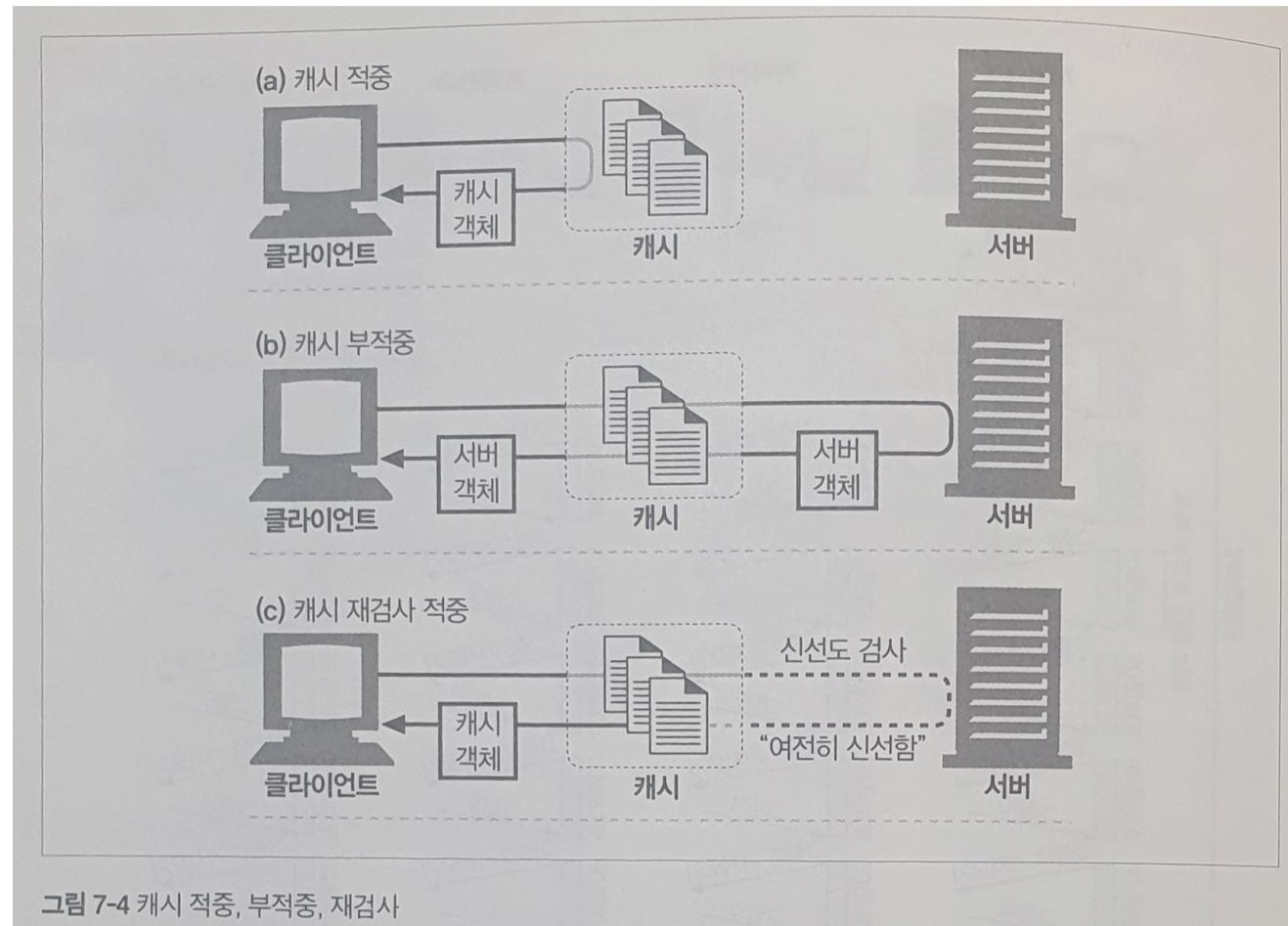
(클라우드플레이어, 구글, 페이스북 등 대형 서비스에서 활발히 사용 중입니다)

- 캐시를 클라이언트 가까운 지점에 배치하면, 문서가 전송되는 거리를 수천 킬로미터에서 수십 미터로 줄일 수 있어 지연을 현저히 완화할 수 있다.
 - **CDN** : 전 세계 여러 지역에 분산된 캐시 서버(노드)들로 구성된 네트워크

7.5 캐시 적중과 부적중

캐시의 한계

- 캐시는 유용하지만, **모든 문서를 저장하지는 않음.**
- 따라서 어떤 요청은 캐시에서 바로 처리되지 못할 수 있음.



요청에 따라서 적중, 부적중, 재검사적중 3가지가 있다.

7.5.1 재검사(Revalidation)

- 캐시는 문서 사본을 보관하고 있으나, 그 사본이 최신인지 확인할 필요가 있다.
- 이 확인 과정을 **재검사(HTTP Revalidation)** 라고 하며, 서버에 직접 확인 요청을 보낸다.
 - 캐시가 사본을 가진 상태에서 서버에 **재검사 요청** 전송
 - 서버는 콘텐츠 변경 여부를 확인하고 세 가지 방식으로 응답:
 1. 재검사 적중 (캐시된 사본이 최신인 경우)
 - 서버는 **HTTP 304 Not Modified** 응답을 보냄
 - 캐시는 사본이 여전히 유효하다고 판단 → **기존 사본 재사용**
 - 장점: 빠르고, 대역폭 절약
 2. 재검사 부적중 (사본이 변경된 경우)
 - 서버는 **콘텐츠 전체와 함께 HTTP 200 OK** 응답을 보냄
 - 캐시는 새로 받은 콘텐츠로 사본을 갱신함
 3. 객체 삭제됨 (서버에서 리소스가 삭제됨)
 - 서버는 **404 Not Found** 응답을 보냄
 - 캐시는 해당 사본을 삭제

7.5.2 적중률

- **적중률(Hit Ratio)**: 캐시가 요청을 직접 처리한 비율
 - 0% : 모두 부적중 (항상 서버에 요청)
 - 100% : 모두 적중 (캐시에서만 응답)
- 캐시 효율성의 주요 지표로, 높을수록 네트워크와 서버 부하 감소

- 캐시 적중률은 캐시 용량, 사용자 습관, 콘텐츠 변경 빈도 등에 따라 달라짐

7.5.3 바이트 적중률

- **바이트 적중률(Byte Hit Ratio)**: 전체 전송된 데이터 양 중 캐시로부터 제공된 비율
- 문서 크기와 무관하게 전체 트래픽 절감 효과를 정확하게 반영
- 적중률이 높을수록 인터넷 대역폭 절약과 비용 감소에 유리

7.5.4 적중과 부적중의 구분

- HTTP는 기본적으로 응답이 캐시 적중인지 구분 불가
- 해결 방법:
 - **Via 헤더**: 응답 경로에 대한 정보 포함
 - **Date 헤더**: 응답 시간 비교로 신선도 판단
 - **Age 헤더**: 응답이 캐시된 시점 기준 나이 판단

7.6 캐시 토플로지

토플로지: 구성 요소들이 어떻게 연결되어 있는지, 또는 배치되어 있는 구조나 형태

7.6.1 개인 전용 캐시 (Private Cache)

- 사용자 개인만 접근 가능 (브라우저 내부 캐시 등)
- 자주 쓰는 페이지를 로컬 디스크나 메모리에 저장
- 예: Chrome의 `about:cache`

7.6.2 공용 프락시 캐시 (Public/Shared Proxy Cache)

- 여러 사용자가 함께 사용하는 캐시
- ISP나 조직에서 운영하는 프락시 서버
- 자주 요청되는 문서를 공유해 트래픽 절감

7.6.3 프락시 캐시 계층들

- 다단계 캐시 계층 구조
 - 가까운 캐시에서 적중하지 않으면 → 상위 캐시로 요청 전달
 -

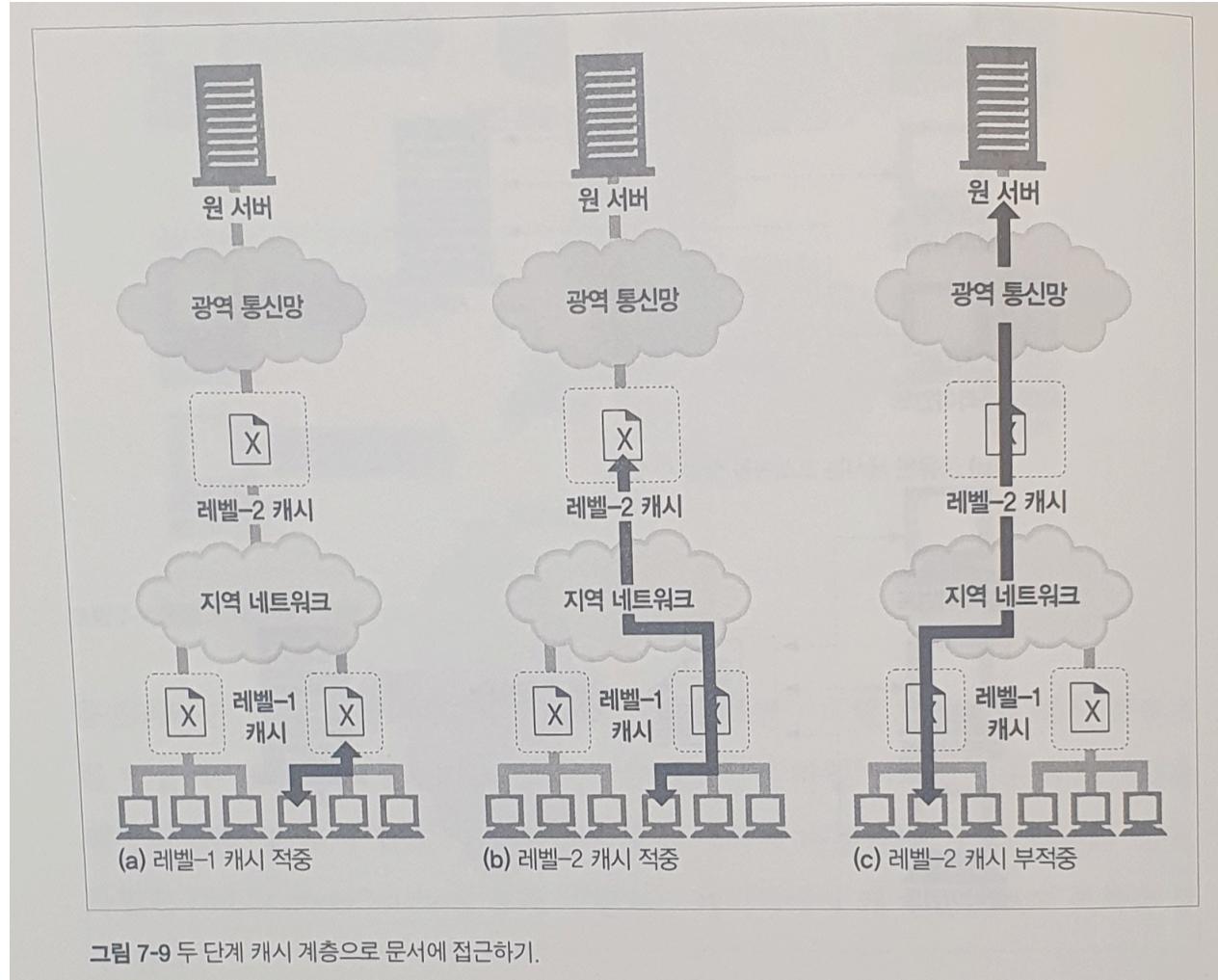


그림 7-9 두 단계 캐시 계층으로 문서에 접근하기.

- 그림 7-9:
 - (a): 1단계 캐시 적중
 - (b): 1단계 부적중 → 2단계 적중
 - (c): 모든 캐시 부적중 → 원 서버로 전달
- 장점: 대역폭 절약, 성능 향상, 트래픽 분산

7.6.4 캐시망, 콘텐츠 라우팅, 피어링

- 단순한 계층 구조가 아니라, **여러 캐시들이 서로 협력하는 복잡한 구조**를 갖춘 캐시망을 사용합니다. 이 캐시망은 각 요청에 대해 **캐시 간 대화**를 통해 어떤 부모 캐시로 갈지, 또는 원 서버로 갈지를 **동적으로 결정**합니다.
- **콘텐츠 라우팅 방식**

캐시들은 다음과 같은 다양한 라우팅 전략을 수행할 수 있습니다:

1. URL 기반 동적 선택

URL에 따라 부모 캐시 또는 원 서버를 동적으로 선택함.

2. URL 기반 고정 선택

URL에 따라 특정 부모 캐시를 고정적으로 선택함.

3. 로컬 캐시 우선

부모 캐시에 가기 전에 먼저 로컬 캐시에 캐시된 사본이 있는지 확인.

4. 다른 캐시의 콘텐츠 허용/차단

다른 캐시에서 일부 콘텐츠에만 접근할 수 있게 허용하거나, 외부 인터넷을 통한 접근(Internet transit)은 차단할 수 있음.

- **형제 캐시(Peer Cache)**

- 여러 조직이 서로의 캐시를 공유하고 상호 검색이 가능하도록 구성한 형태.
- 이를 **형제 캐시**라고 부르며, **선택적 피어링(peer relationship)**을 통해 설정.
- 그림 7-10은 조직 A와 조직 B가 서로의 캐시를 공유하는 예시를 보여줌.

- **형제캐시 지원 프로토콜**

- HTTP는 기본적으로 형제 캐시를 지원하지 않음.
- 대신 다음과 같은 캐시 프로토콜을 사용함:

- ICP (Internet Cache Protocol)

- HTCP (Hyper Text Cache Protocol)

이 프로토콜들은 HTTP에 기반해 확장되어 캐시 간 커뮤니케이션을 가능하게 함.

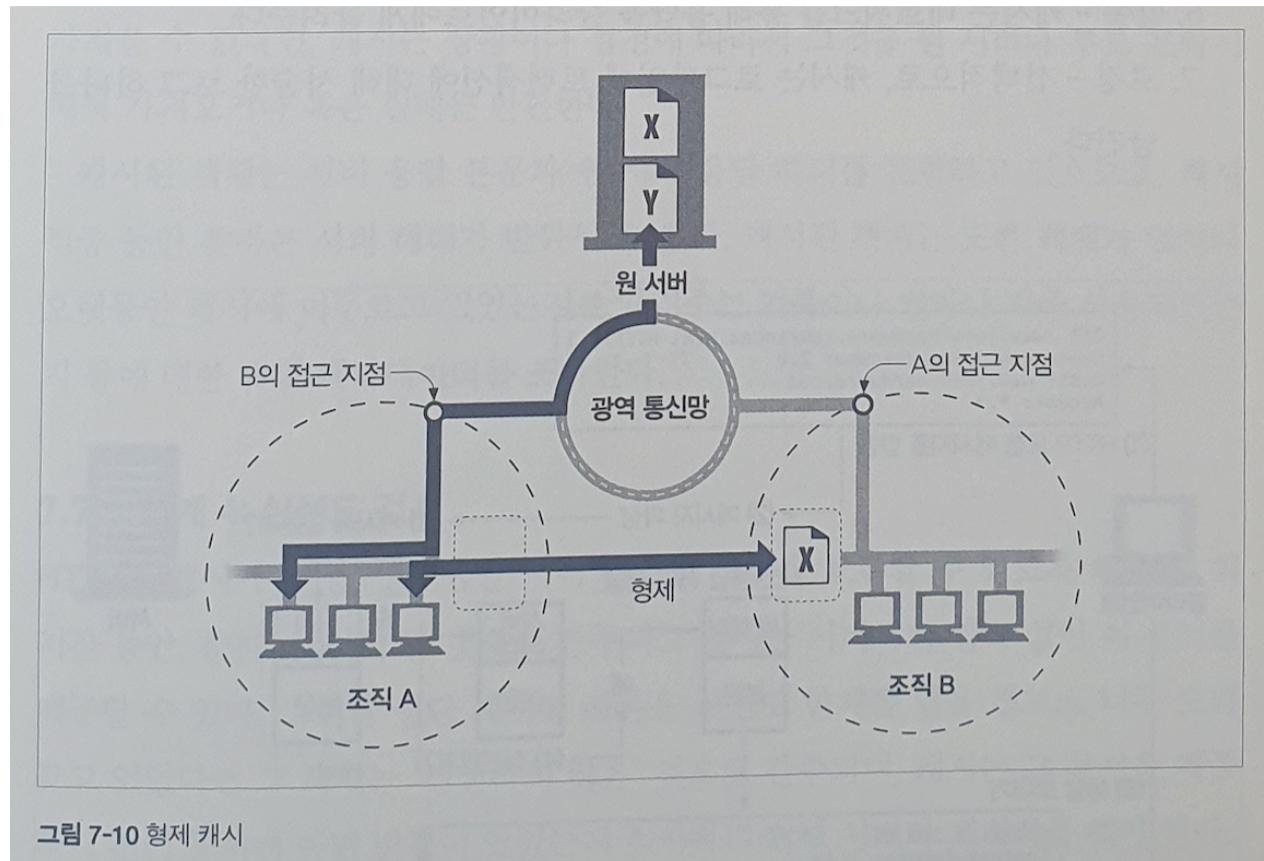


그림 7-10 형제 캐시

- “원서버”는 x, y 컨텐츠 보유
- “광역통신망”은 원서버와 조직(조직A, 조직B) 간의 통신을 연결
- 조직 A, 조직 B는 회사와 기간 즉, 사용자
- X 캐시 서버 (조직 B) : 콘텐츠를 캐시하는 서버로, 조직 B에 위치한 상황.
- 형제(Peer) 캐시 연결
 - 조직 A는 직접 원 서버에 요청하는 대신, 조직 B의 캐시 서버(X)에 먼저 요청.
 - 이때 두 조직 간에는 형제 관계(peer relationship)가 설정되어 있어 캐시를 공유.

데이터 흐름 해석 시나리오

1. 조직 A의 사용자가 콘텐츠(X or Y)를 요청합니다.
2. 조직 A의 로컬 캐시에 콘텐츠가 없을 경우:
 - 조직 A는 먼저 **조직 B의 캐시 서버(X)**에게 요청합니다. (→ 형제 요청)
3. 조직 B의 캐시 서버(X)가 캐시를 가지고 있다면 응답을 보내고,
4. 없다면 그제서야 원 서버로 요청이 전달됩니다.

복잡한 캐시망에서는 정적 계층 구조 대신, URL과 콘텐츠, 캐시 상황에 따라 라우팅 방식을 전환하여 효율을 높히도록 한다.

7.7 캐시 처리 단계 요약

웹 캐시는 HTTP 요청 처리 시 다음 7단계 절차를 거쳐 응답을 합니다.

캐시는

요청 분석 → 사본 탐색 → 신선도 확인 → 응답 생성 및 전송 → 로깅

의 흐름으로 작동한다.

7.7.1~7.7.7

- 요청받기, 파싱, 검색, 신선도검사, 응답생성, 전송, 로깅

1. 요청 받기 (Request Reception)

- 네트워크로부터 요청 메시지를 수신
- 고성능 캐시는 커넥션 감시 및 병렬 처리도 수행

2. 파싱 (Parsing)

- 요청 메시지를 분석해 URL, 헤더 등의 정보 추출
- 헤더 필드를 조작하거나 캐시 정책 판단에 사용

3. 검색 (Lookup)

- 캐시된 사본이 존재하는지 검색
- 메모리, 디스크, 혹은 원격 캐시에서 찾음
- 캐시된 객체에는 원 서버 응답 헤더 및 메타데이터 포함됨

4. 신선도 검사 (Freshness Check)

- 사본이 유효 기간 내이면 바로 제공
- 오래되었거나 만료된 경우 서버에 **재검사 요청**
 - 성공 시: 304 Not Modified → 캐시 갱신
 - 실패 시: 새 콘텐츠 수신 → 캐시 갱신

5. 응답 생성 (Response Construction)

- 원 서버 응답처럼 보이도록 응답 헤더 생성
- 캐시 관련 헤더 삽입:
 - Cache-Control, Age, Expires 헤더, Via, Date 등

6. 전송 (Delivery)

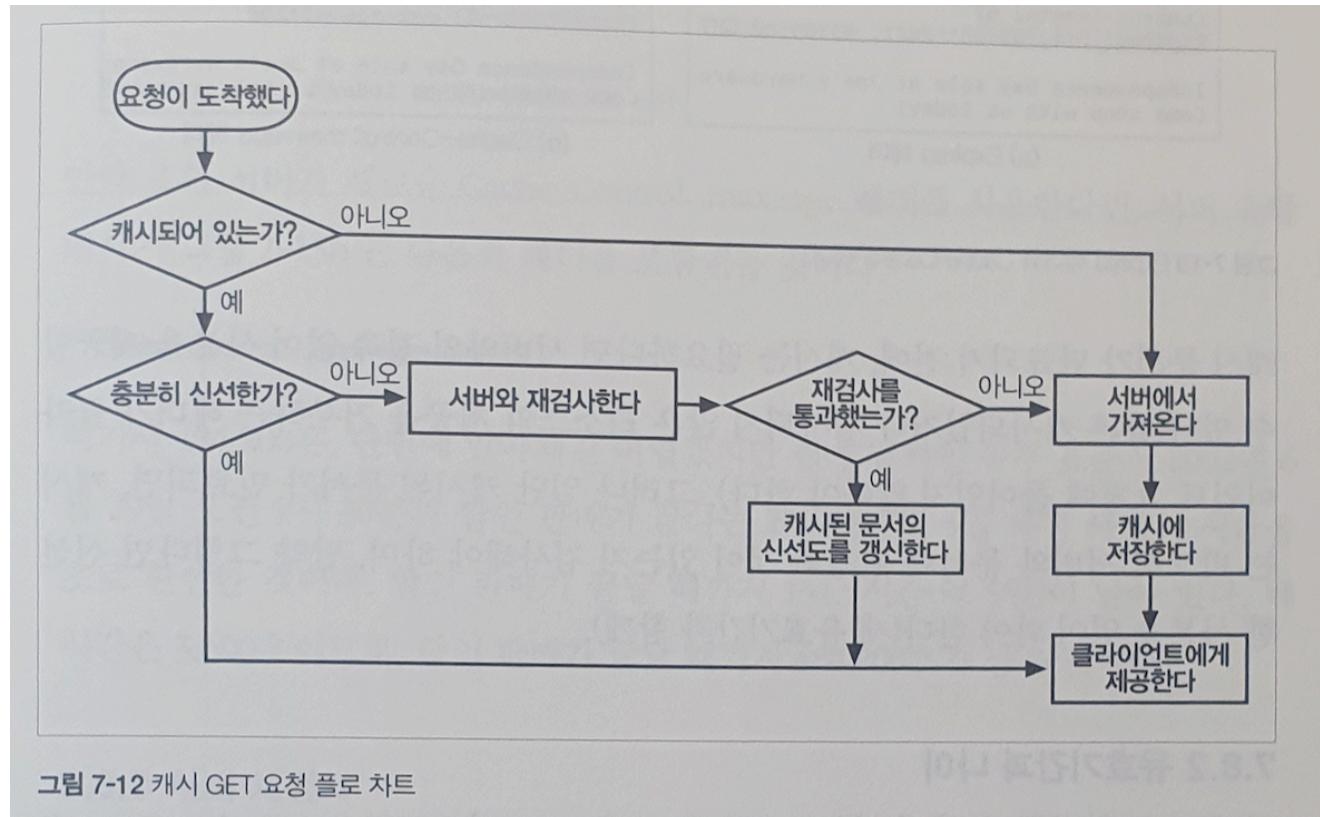
- 생성한 응답을 클라이언트에게 전달
- 고성능 캐시는 I/O 병목을 줄이기 위해 메모리-네트워크 최적화 적용

7. 로깅 (Logging)

- 트랜잭션 기록 및 통계용 로그 작성:
 - 요청 URL, 응답 상태, 적중/부적중 여부 등
- 대표적인 포맷:
 - Squid log format, Netscape extended common log format

7.7.8 캐시 처리 플로 차트 (그림 7-12)

- 캐시 GET 요청 순서도



7.8 사본을 신선하게 유지하기

- 캐시된 사본은 서버의 원본 문서와 항상 일치하지 않음.
- 문서는 시간이 지나면서 변하고, 캐시도 그에 따라 신선도(freshness)를 관리해야 함.

7.8.1 문서 만료 (Document Expiration)

- 서버는 `Cache-Control`, `Expires` 헤더를 통해 문서의 **유효기간**을 지정할 수 있음.
- 유효기간이 지나지 않으면 캐시는 **서버에 재접속하지 않고 응답 가능**.
- `Cache-Control: max-age` 는 문서 수명(초 단위)을 지정.
- `Expires` 는 절대시간(날짜 형식)으로 지정.
- 두 헤더 중 `Cache-Control` 이 우선됨.

7.8.2 유효기간과 나이 (Age)

- `max-age`는 문서가 처음 생성된 시점 기준 최대 유효 시간.
- `Age`는 캐시 내에서 경과된 시간.
- 클라이언트는 `max-age - Age` 계산으로 신선한지 판단 가능.

7.8.3 서버 재검사 (Revalidation)

- 문서 만료 후에는 서버와 **재검사** 필요.
- 304 Not Modified 응답 → 캐시 유지하며 헤더만 갱신.
- 200 OK 응답 + 새 본문 → 문서가 변경된 경우 새로 저장.

7.8.4 조건부 요청과 재검사

- 조건부 메서드는 HTTP 캐시에서 **효율적인 재검사를** 가능하게 합니다.
- 클라이언트(또는 프록시)는 서버에게 "이 문서가 여전히 유효한가?"를 확인하기 위해 **조건부 GET 요청**을 보냅니다.
- 웹서버는 조건이 참인경우에만 객체를 반환해준다. → 불필요한 네트워크 사용량 줄이는 효과를 준다.
- 캐시 재검사에 사용되는 두 조건부헤더는 아래와 같다.

헤더	설명
If-Modified-Since: <date>	만약 문서가 주어진 날짜 이후로 수정되었다면 요청 메서드를 처리한다. 이것은 캐시된 버전으로부터 콘텐츠가 변경된 경우에만 콘텐츠를 가져오기 위해 Last-Modified 서버 응답 헤더와 함께 사용된다.
If-None-Match: <tags>	마지막 변경된 날짜를 맞춰보는 대신, 서버는 문서에 대한 일련번호와 같이 동작하는 특별한 태그(부록 C의 "ETag"를 보라)를 제공할 수 있다. If-None-Match 헤더는 캐시된 태그가 서버에 있는 문서의 태그와 다를 때만 요청을 처리한다.

7.8.5 If-Modified-Since: 날짜 재검사

- 가장 흔한 조건부 헤더.
- 클라이언트가 가진 리소스가 서버에서 마지막으로 수정된 이후 변경되었는지를 검사하기 위해 사용된다
이를 확인하기 위해서 IMS 요청을 한다. (조건부 GET 요청)
- 서버는 해당 날짜 이후 문서가 변경되었는지를 확인한다.
 - 만약, 변경되었으면:** 조건이 충족되어 200 OK 응답과 함께 새로운 문서를 보냄.
 - 만약, 변경되지 않았으면:** 조건이 거짓이므로 304 Not Modified 응답을 보냄 (본문은 없음).
- 헤더의 동작 특성
 - 서버는 일반적으로 Last-Modified 헤더를 통해 문서의 최근 변경 일시를 제공하며, If-Modified-Since와 함께 사용된다.
 - 일부 서버는 날짜 비교 대신 문자열 비교 방식으로 "이 날짜 이후에 변경되었는가?"가 아니라 "이 날짜가 마지막 수정일인가?"를 확인하는 방식으로 처리하기도 한다.

7.8.6 If-None-Match: 엔터티 태그 재검사

If-None-Match 조건부 요청 헤더는 **ETag(Entity Tag)** 값을 기준으로 캐시된 자원이 여전히 유효한지 확인할 때 사용됩니다. 이는 If-Modified-Since 헤더가 적용되기 어려운 경우에 효과적입니다.

- 사용하는 경우
 - 백그라운드 프로세스로 문서가 주기적으로 동일하게 재작성될 때
 - 문서 내용은 같지만 헤더(예: 날짜)만 변경된 경우
 - 서버가 정확한 변경 시점을 추적할 수 없을 때
 - 1초보다 짧은 간격으로 갱신되는 문서의 경우
- 순서
 - 캐시는 서버에 If-None-Match 헤더로 자신의 ETag 값을 전달합니다.
 - 서버는 현재 문서의 ETag와 비교하여 동일하면 **304 Not Modified**를 응답하고, 다르면 **200 OK**와 함께 새로운 콘텐츠와 ETag를 반환합니다.
- 예시
 - 클라이언트가 ETag: "v2.6" 값을 가진 캐시를 보유한 상태에서 서버에 요청을 보냄
 - 서버 문서도 "v2.6"이면 304 응답
 - 서버 문서가 "v3.0" 등으로 변경되었다면 200 응답과 새로운 ETag 포함

7.8.7 약한 검사기와 강한 검사기

http는 서버가 가지고 있는 캐시정보와 클라이언트가 가지고 있는 캐시정보가 동일한지를 판단하기 위해 ETAG 와 Last-Modified정보를 확인하는데,,,

여기서 "강한검사기", "약한검사기" 를 통해서 캐시의 변동여부를 판단한다.

** ETag : Entity Tag (리소스 버전 태그)

1. 강한 검사기 (Strong Validator)

- 콘텐츠가 바뀔 때마다 엔터티 테그(ETag)가 변경된다.
- 콘텐츠의 의미 있는 변경이 있을 때만 엔터티 테그(ETag)가 바뀐다.
- 캐시 유효성 검사 시, 정확한 일치를 요구한다.
- 서버는 강한 검사기 값을 재사용하면 안 된다.

2. 약한 검사기 (Weak Validator)

- 콘텐츠가 조금 변경되었더라도, 서버가 "거의 같다"고 판단하면 같은 엔터티 테그(ETag)를 유지할 수 있다.
- 유의미한 변경이 아닌 경우에도 같은 엔터티 테그(ETag)를 사용할 수 있다.
- HTTP/1.1은 약한 검사기를 지원하며, 엔터티 테그(ETag) 앞에 **W/** 접두사를 붙여 구분한다.

상황에 따라서 서버는 강한검사기와 약한검사기를 사용하여 리소스를 제공하여 캐시를 유연하게 적용할 수 있도록 할 수 있다.

7.8.8 언제 엔터티 태그를 사용하고 언제 Last-Modified를 사용하는가?

- **HTTP/1.1 클라이언트**는 서버가 **ETag(엔터티 태그)** 를 반환하면 그 **ETag**를 반드시 사용하여 재검사해야 한다.
- 서버가 **Last-Modified** 값만 반환한다면, 클라이언트는 **If-Modified-Since** 헤더를 사용하여 재검사 요청 가능.

서버 상황에 따라서 Entity tag(ETag) 와 Last-modified Tag 가 모두 전송가능한 상황이라면,

→ 둘 다 보내주는것이 좋고,

원 서버가 ETag를 보낼 수 없을 경우에는

⇒ 약한 ETag라도 제공하거나

⇒ **Last-Modified** 값만 보내는 것이 좋다.

요약하자면, 캐시는 서버가 제공하는 Expires나 Cache-Control 헤더를 통해 문서가 여전히 유효한지 판단하고, 만료된 경우에는 조건부 요청으로 서버와 재검사를 하여 최신 사본을 유지합니다.

7.9 캐시 제어 (Cache Control)

HTTP는 문서의 만료 시점과 캐시 유지 기간을 제어하기 위해 다양한 헤더를 제공함.

주요 캐시 제어 헤더

- **Cache-Control: no-store** : 캐시 저장 자체 금지.
- **Cache-Control: no-cache** : 캐시는 가능하나, 재검증 필요.
- **Cache-Control: must-revalidate** : 만료된 문서는 반드시 원 서버 재검증 필요.
- **Cache-Control: max-age** : 최대 생존 시간(초).
- **Expires** : 절대 만료 날짜(시간 동기화 이슈로 잘 안씀).
- 캐시가 설정이 없으면 휴리스틱 만료로 추정.

7.9.1 no-cache와 no-store 응답 헤더

- **no-store** : 캐시에 저장하지 않음. 민감한 데이터를 다룰 때 사용.
- **no-cache** : 캐시는 가능하되, 매 요청 시 원 서버 재검증이 필요.

- `Pragma: no-cache` : HTTP/1.0 호환성을 위한 과거 방식.

7.9.2 Max-Age 응답 헤더

- 서버에서 문서가 전송된 이후 몇 초간 캐시가 유효한지를 정의한다.
- `s-maxage` 는 공유 캐시에만 적용된다.
- `max-age=0` 을 설정하면(maximum aging, 최대 나이먹음)을 설정하여 , 매 요청마다 재검증을 강제할 수 있다.

7.9.3 Expires 응답 헤더

>Expires = 만료 시각

- 더이상 권장하지 않는 방식이다.
- 초단위의 시간대신 절대 시간으로(실제 만료시간) 문서 만료 시점을 명시한다.
- 클라이언트와 서버 간 시계 차이로 인해 신뢰성이 낮아 현재는 권장되지 않으며, 대신 `max-age` 사용을 권장한다.

7.9.4 must-revalidate 응답 헤더

- 캐시가 만료된(신선하지 않은) 콘텐츠를 원 서버에 재검사 없이 제공해서는 안 된다는 의미입니다.
- 반드시 원 서버에 **재검사(revalidation)**를 거쳐야 하며
- 재검사가 불가능한 상황 (예: 원 서버가 다운됨)에는 절대로 캐시된 사본을 제공하지 않고
- 재검사가 불가능할 경우에는 **504 Gateway Timeout** 오류를 반환해야 합니다.

7.9.6 클라이언트 신선도 제약

클라이언트(브라우저)는 캐시된 콘텐츠가 신선하지 않다고 판단되면, 이를 강제로 갱신시키기 위한 수단을 제공한다. 대표적으로 "새로 고침(리프레시)" 버튼이 이에 해당하며, 이 버튼을 누르면 추가적인 **Cache-Control** 요청 헤더를 포함한 GET 요청이 발생하고, 서버에 재검사를 요청하게 된다. 이 과정에서 기존 캐시를 무시하고 콘텐츠를 갱신해버린다.

- 클라이언트는 Cache-Control 요청 헤더를 활용하여 서버에 **만료 제약**을 엄격히 요구할 수 있다.
- 특히 애플리케이션에서 문서를 항상 최신 상태로 유지하려는 경우(예: 새로 고침 버튼 또는 실시간 정보 앱)에는 이러한 제약을 클라이언트가 강하게 설정할 수 있다.

하지만 이렇게 클라이언트가 신선도를 직접 제어하는 방식은 **성능 저하, 신뢰성 문제, 비용 증가** 등의 부작용이 있으므로, 필요한 경우에만 사용해야 하며 신중하게 적용되어야 한다.

7.9.7 주의할 점

- 퍼블리셔가 너무 길거나 짧은 만료 시간을 설정할 경우 캐시가 잘못된 판단을 할 수 있다.
- 명확한 만료 기준을 설정하지 않으면 캐시는 신선도를 부정확하게 판단할 수 있다.

(유효기간을 길게 설정할경우(신선) : 최초정보가 캐싱되어 영원히 신규정보가 반영되지 않을수 있다.)

(유효기간을 짧게 설정할경우(비신선) : 정보변경이 없음에도 캐싱이 무조건 최신화 될 수 있다.)

7.10 캐시 제어 설정

웹 서버는 HTTP 헤더를 통해 문서의 **만료와 캐시 제어**를 설정할 수 있는 여러 메커니즘을 제공합니다.

- ▼ 7.10.1 아파치로 HTTP 헤더 제어하기, 7.10.2 HTTP-EQUIV를 통한 HTML 캐시 제어 (한번씩 보세요)

7.10.1 아파치로 HTTP 헤더 제어하기

아파치 웹 서버는 다양한 모듈을 통해 캐시 제어가 가능하며, 대표적으로 다음과 같은 모듈이 있습니다.

- **mod_headers**

설정 파일 내에서 HTTP 헤더를 직접 설정할 수 있으며, 특정 파일 유형에 대해 헤더를 붙이는 필터 역할도 가능.

예:

```
apache
CopyEdit
<Files *.html>
  Header set Cache-control no-cache
</Files>
```

- **mod_expires**

`Expires` 헤더를 자동 생성하는 모듈. 문서의 마지막 접근 시점이나 수정 시점을 기준으로 만료 시간 설정 가능.

예:

```
nginx
CopyEdit
ExpiresDefault A3600
ExpiresDefault "access plus 1 week"
```

- **mod_cern_meta**

파일별로 HTTP 헤더 정보를 메타파일로 지정할 수 있도록 지원.

7.10.2 HTTP-EQUIV를 통한 HTML 캐시 제어

HTML 문서의 `<meta>` 태그에 `HTTP-EQUIV` 속성을 이용하여 HTTP 응답 헤더처럼 캐시 제어 정보를 포함할 수 있음.

예:

```
html
CopyEdit
<META HTTP-EQUIV="Cache-control" CONTENT="no-cache">
```

문제점:

- HTTP 서버가 해당 태그를 해석하지 않거나, 명시된 헤더를 응답에 포함하지 않는 경우가 많음.
- 일부 브라우저는 이 태그를 실제 HTTP 헤더로 오해하고 적용함.
- 프락시 캐시와 충돌할 수 있어 혼란 발생 가능.

결론:

HTML 문서의 캐시 제어는 **HTTP 응답 헤더**를 통해 설정하는 것이 가장 확실한 방법이며, `<meta>` 태그 방식은 제한적 환경에서만 고려할 수 있음.

7.11 자세한 알고리즘

pass.

7.12 캐시와 광고

캐시와 광고 시스템이 상호 충돌할 수 있는 지점을 설명하며, 콘텐츠 제공자와 광고 회사, 그리고 퍼블리셔들이 캐시에 대해 어떻게 대응하는지 확인해보자.

7.12.1 광고 회사의 딜레마

- **장점**
 - 콘텐츠 제공자 입장에선 캐시가 있으면 서버 부담과 네트워크 비용이 줄고, 콘텐츠가 더 빠르게 사용자에게 전달되어 사용자 경험도 좋아짐.
- **문제점:**

- 많은 콘텐츠 제공자들은 광고 노출 수에 따라 수익을 얻는데, 캐시에 의해서 광고 카운트를 할 수 없게되면 실제 서버 접근이 줄어들어 광고 노출이 정확히 측정되지 않음.
- 결과:
광고 수익이 줄어드는 상황이 되므로, 광고 회사 입장에서는 캐시를 꺼려할 수 있음.

7.12.2 퍼블리셔의 응답

- 퍼블리셔의 대응
 - 캐시 무력화 기법을 사용하여 광고시청수 카운트를 확실하게 올리기로 한다.
 - 이로서 시청수를 명확하게 카운트 할 수 있게되었으나, 트래픽이 올라가는 결과를 주게되었다.
 - 이상적으로는 캐시가 퍼블리셔 트래픽을 흡수한 뒤, 얼마나 광고 노출이 발생했는지 정확히 알려줄 방법이 필요하다.

▼ 해설

1. “캐시가 퍼블리셔 트래픽을 흡수한 뒤”

- 캐시는 클라이언트가 같은 리소스를 요청할 때 원 서버(프론트엔드 서버의 데이터)까지 요청이 가지 않도록 중간에서 응답을 대신하는 역할을 한다.
- 이 말은 즉, 사용자가 웹 페이지나 광고를 요청했을 때, 이 요청이 프론트엔드의 원 서버로 가지 않고 캐시 서버에서 처리되었다는 뜻.
- 즉, 퍼블리셔의 트래픽이 줄어들게 되는 것을 의미합니다.

2. “얼마나 광고 노출이 발생했는지”

- 퍼블리셔는 광고를 **몇 번 보여줬는지(노출 수)**가 중요해요.
그래야 광고 수익을 정산할 수 있기 때문이죠.

3. “정확히 알려줄 방법이 필요하다”

- 캐시가 트래픽을 흡수하게 되면, 퍼블리셔는 실제로 사용자에게 광고가 몇 번 노출됐는지를 정확히 모르게 됩니다.
- 왜냐면 HTTP 요청이 퍼블리셔까지(프론트서버) 도달하지 않기 때문.
 - (프론트서버까지 트래픽이 도달해야 카운트로직을 탈것인데 도달하지 않으면 카운트할 방법이 없다.)
- 그래서 퍼블리셔는 캐시가 사용자에게 광고를 보여줬다면, 그것도 포함해서 '몇 번 광고가 노출되었는지'를 정확히 알 수 있는 기술적인 방법이 필요하다는 뜻.
- 캐시를 무력화 하는 방법으로는 광고 접근을 확인할 수 있는 방법이지만, 캐시의 긍정적인 효과가 사라진다. (트래픽이 온전히 발생)
- 그래서 이상적인 해결책이 필요하다
 - 캐시를 무력화하지 않고도 광고 노출 수를 정확히 파악할 수 있는 방법이 필요함.

정리 :

- 캐시는 트래픽을 줄여주는 장점이 있음.
- 그러나 광고 수익 측정에는 방해가 되므로, 캐시를 무력화하게 됨.
- 이 **충돌을 해결할 수 있는 방법(캐시는 유지하면서 정확한 시청 수 측정)**이 이상적이라는 의미입니다.

해결책 2가지

- 7.12.3 로그 마이그레이션
- 7.12.4 적중 측정과 사용량 제한

7.12.3 로그 마이그레이션

이상적 해결책: 캐시가 원 서버로 요청을 보내지 않도록 하고, 그 대신 적중 로그(hit log)를 유지.

문제점:

- 로그를 수동으로 전달하여 카운트하고 처리해야하며, 로그 용량도 커서 옮기기 어렵다.

- 서비스별 표준화가 되지 않아 상황에 따라 다르게 적용되어야 할 수 있다.
- 로그전달 및 카운트 처리시 프라이버시 문제가 존재.(처리중 유출 가능성 존재)

7.12.4 적중 측정과 사용량 제한

- **RFC 2227**에서 정의된 간단한 방식:
 - Meter라는 새로운 HTTP 헤더를 추가하여, 특정 URL의 **캐시 적중 횟수**를 서버에 정기적으로 보고.
- **장점:** 콘텐츠를 다시 가져오지 않고도 **사용량 통계만 수집** 가능.
- **추가 기능:** 서버는 캐시에 대해
 - "언제까지 몇 번까지만 문서를 제공하라"는 식의 **제한을 둘 수 있음**.
 - 이를 **사용량 제한(usage-limiting)** 이라 부름.
- 캐시 리소스가 너무 많이 사용되기 전에 서버가 **미리 통제**할 수 있음.