9장 - 웹 크롤러 설계

- 웹 크롤러는 로봇(robot) 또는 스파이더(spider)라고도 불리며 검색 엔진에서 널리 쓰인다.
  - 주 목적으로는 웹에 새로 올라오거나 갱신된 콘텐츠를 찾아내는 것이 주된 목적이다.
  - 콘텐츠는 웹 페이지 일 수도 있고, 이미지나 비디오, PDF 파일일 수도 있다.
- 웹 크롤러는 웹 페이지에서 시작하여 링크를 따라 다니면서 새로운 콘텐츠를 수집한다.

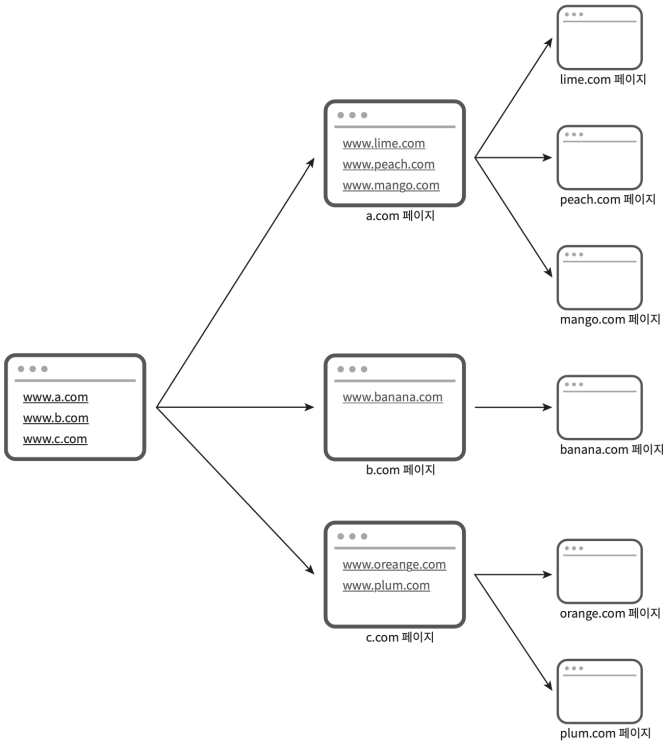

그림 9-1

웹 크롤러의 동작 방식

다양한 크롤러 예제

- **검색 엔진 인덱싱(search engine indexing)**
  - 크롤러의 가장 보편적인 용례
  - 크롤러는 웹 페이지를 모아 검색 엔진을 위한 로컬 인덱스(local index)를 만든다.

    - 일례로 Googlebot은 구글(Google) 검색 엔진이 사용하는 웹 크롤러다.
- **웹 아카이빙(web archiving)**
    - 나중에 사용할 목적으로 장기보관하기 위해 웹에서 정보를 모으는 절차
    - 많은 국립 도서관이 크롤러를 돌려 웹 사이트를 아카이빙하고 있다.
        - ex) 미국 국회 도서관(US Library of Congress), EU 웹 아카이브
- **웹 마이닝(web mining)**
    - 웹의 폭발적 성장세는 데이터 마이닝(data mining) 업계에 전례 없는 기회이다.
    - 웹 마이닝을 통해 인터넷에서 유용한 지식을 도출해 낼 수 있는 것이다.
    - 금융 기업은 크롤러를 사용해 주총 자료나 연차 보고서를 다운받아 기업의 사업 방향을 알아내기도 한다.
- **웹 모니터링(web monitoring)**
    - 크롤러를 사용하면 인터넷이나 저작권이나 상표권이 침해되는 사례를 모니터링할 수 있다.
    - 일례로 디지마크(Digimarc)사는 웹 크롤러를 사용해 해적판 저작물을 찾아내서 보고한다.

> 웹 크롤러의 복잡도는 웹 크롤러가 처리해야 하는 데이터의 규모에 따라 달라진다.
> 별도의 엔지니어링 팀을 꾸려 지속적으로 관리하고 개선해야 하는 대형 프로젝트일 수도 있고,
> 몇 시간이면 끝내는 작은 프로젝트 일 수도 있다.
> 따라서
> **설계할 웹 크롤러가 감당해야 하는 데이터의 규모와 기능들을 알아내야** 한다.

1단계 : 문제 이해 및 설계 범위 확정

> ### 웹 크롤러의 기본 알고리즘

1. URL 집합이 입력으로 주어지면, 해당 URL들이 가리키는 모든 웹 페이지를 다운로드한다.
2. 다운받은 웹 페이지에서 URL들을 추출하낟.
3. 추출된 URL들을 다운로드할 URL 목록에 추가하고 위의 과정을 처음부터 반복한다.

지원자: 이 크롤러의 주된 용도는 무엇인가요? 검색 엔진 인덱스 생성용 인가요? 아니면 데이터 마이닝? 아니면 그 외의 다른 용도가 있나요?

면접관: 검색 엔진 인덱싱에 쓰일 것입니다.

지원자: 매달 얼마나 많은 웹 페이지를 수집해야 하나요?

면접관: 10억(1billion)의 웹 페이지를 수집해야 합니다.

지원자: 새로 만들어진 웹 페이지나 수정된 웹 페이지도 고려해야 하나요?

면접관: 그렇습니다.

지원자: 수집한 웹 페이지는 저장해야 합니까?

면접관: 네. 5년간 저장해 두어야 합니다.

지원자: 중복된 콘텐츠는 어떻게 해야 하나요?

면접관: 중복된 콘텐트를 갖는 페이지는 무시해도 됩니다.

기능 요구사항과 더해 좋은 웹 크롤러가 만족시켜야 할 속성

- **규모 확장성**
  - 오늘날 웹에는 수십억 개의 페이지가 존재한다.
  - 따라서 **병행성(parallelism)을 활용**하면 보다 효과적으로 웹 크롤링을 할 수 있다.
- **안정성(robustness)**

    - 잘못 작성된 HTML, 아무 반응이 없는 서버, 장애, 악성 코드가 붙어 있는 링크가 좋은 예다.
    - 크롤러는 이런 비정상적 입력이나 환경에 잘 대응할 수 있어야 한다.
- **예절(politeness)**
    - 크롤러는 수집 대상 웹 사이트에 짧은 시간 동안 너무 많은 요청을 보내서는 안 된다.
- **확장성(extensibility)**
    - 새로운 형태의 콘텐트를 지원하기가 쉬워야 한다.
    - ex) 이미지 파일도 크롤링하고 싶을 때 이를 위해 전체 시스템을 새로 설계해야 한다면 곤란하다.

개략적 규모 추정

> 다음의 추정치는 많은 가정으로 나온 것이다. 따라서 면접관과 합의해 두는 것이 중요하다.

- 매달 10억 개의 웹 페이지를 다운로드한다.
- QPS = 10억(1billion) / 30일 / 24시간 / 3600초 = 대략 `400페이지/초`
- 최대(Peak) QPS = 2 * QPS = 800
- 웹 페이지의 크기 평균은 500k라고 가정
- 10억 페이지 * 500k = 500TB/월.
- 1개월치 데이터를 보고나하는 데는 500TB, **5년간 보관**한다고 가정하면 결국 500TB * 12 * 5 = `30PB`

2단계 : 개략적 설계안 제시 및 동의 구하기

- 다음은 여기과 여기를 참고한 것이라고 함

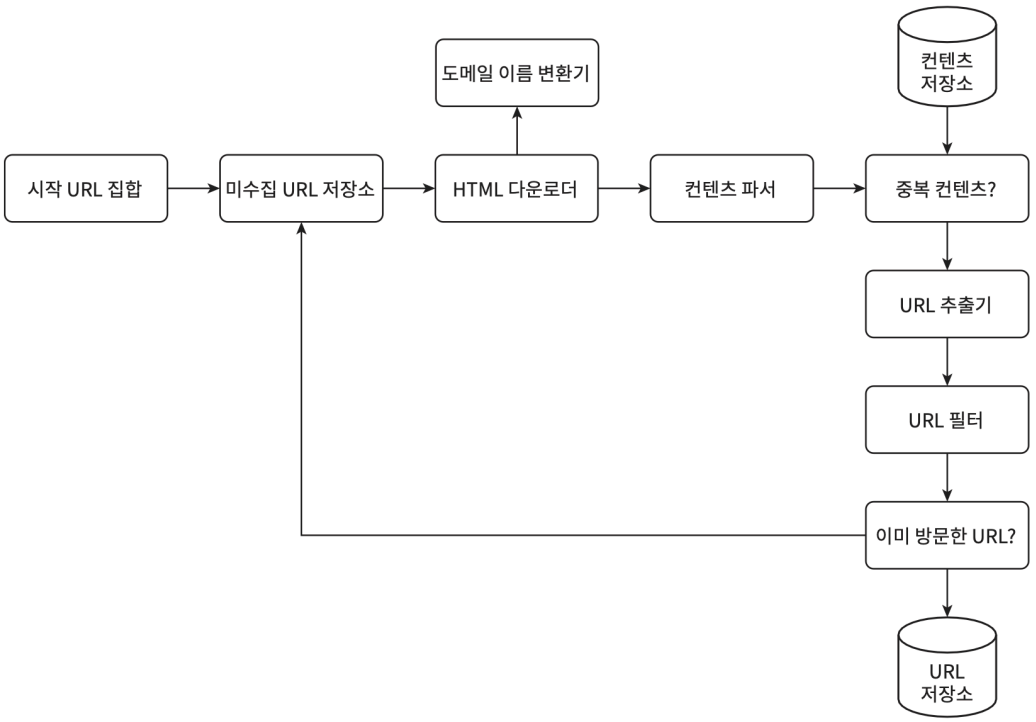

그림 9-2

> 각 컴포넌트를 살펴보면 다음과 같다.

시작 URL 집합

- 웹 크롤러가 시작하는 출발점
  - ex) 어떤 대학 웹사이트로부터 찾아 나갈 수 잇는 모든 웹페이즈를 크롤링하는 가장 직관적인 방법

    해당 대학의 도메인 이름이 붙은 모든 페이지의 URL을 시작 URL로 사용하는 것이다.
  - 전체 웹을 크롤링해야 하는 경우 시작 URL을 고를 때 좀 더 창의적으로 해야 된다.
    - 크롤러가 가능한 한 많은 링크를 탐색할 수 있도록 하는 URL을 고르는 것이 좋다.
- 일반적으로 전체 URL 공간을 작은 부분집합으로 나누는 전략을 사용한다.
  - 나라별로 인기 있는 웹 사이트가 다르난 점에 착안해야 한다.
- 다른 방법으로는 주제별로 다른 시작 URL을 사용해도 좋다.
  - ex) URL 공간을 쇼핑, 스포츠, 건강 등등 주제별로 세분화하고, 각각의 시작 URL 사용

- 시작 URL을 무엇을 쓸 것이냐는 정답이 없고, 면접관도 완벽한 답안을 기대하지 않는다.
  의도가 무엇인지만 정확히 전달하자.

미수집 URL 저장소

- 현대적 웹 크롤러는 크롤링 상태를 `다운로드할 URL`, `다운로드된 URL`의 **두 가지로 나눠 관리**한다.
- 이 중 `다운로드할 URL`을 저장 관리하는 컴포넌트를 **미수집 URL 저장소(URL frontier)**라고 부른다.
  - FIFO 큐라고 생각하면 됨

HTML 다운로더

- 인터넷에서 웹 페이지를 다운로드하는 컴포넌트
  - 다운로드할 페이지의 URL은 미수집 URL 저장소가 제공

도메인 이름 변환기

- 웹 페이지를 다운받으려면 URL을 IP 주소로 변환하는 절차가 필요하다.
- HTML 다운로더는 도메인 이름 변환기를 사용하며 URL에 대응되는 IP 주소를 알아낸다.

콘텐츠 파서

- 웹 페이지를 다운로드하면 파싱(parsing)과 검증(validation) 절차를 거쳐야 한다.
  - 이상한 웹 페이지는 문제를 일으킬 수 있는데다 저장 공간만 낭비하는 상황
- 크롤링 서버 안에 콘텐츠 파서를 구현하면 크롤릴 과정이 느려지게 될 수 있으므로 독립된 컴포넌트로 구성

중복 컨텐츠인가?

- 공개된 연구에 따르면 **29% 가량의 웹 컨텐츠는 중복**이다. (같은 콘텐츠를 여러 번 저장할 수 있음)
- 이 문제를 해결하기 위한 자료 구조를 도입해 데이터 중복과 데이터 처리에 시간을 줄인다.
- 두 HTML 문서를 비교하는 방법으로 두 문서를 문자열로 보고 비교하는 방법이 있지만 성능이 너무 안 좋다.
  비교 대상 무선의 수가 10억인데, 느리고 비효율적이다. 따라서
  효과적인 방법은 해시 값을 비교하는 것이다.

콘텐츠 저장소

- HTML 문서를 보관하는 시스템
- 저장소를 구현하는 데 쓰일 기술을 고를 때는 저장할 데이터의 유형, 크기, 저장소 접근 빈도, 데이터의 유효 기간 등을 종합적으로 고려해야 한다.
- 이 책의 설계안은 **디스크와 메모리를 동시에 사용**하는 저장소를 선택한다.
  - 데이터 양이 너무 많으므로 **대부분의 컨텐츠는 디스크**에 저장
  - **인기 있는 컨텐츠**만 **메모리에** 두어 접근 지연시간을 줄일 것

URL 추출기

- URL 추출기는 HTML 페이지를 파싱하여 링크들을 골라내는 역할을 한다.
- 상대 경로(relative path)는 전부 https://en.wikipedia.org를 붙여 절대 경로(absolute path)로 변환

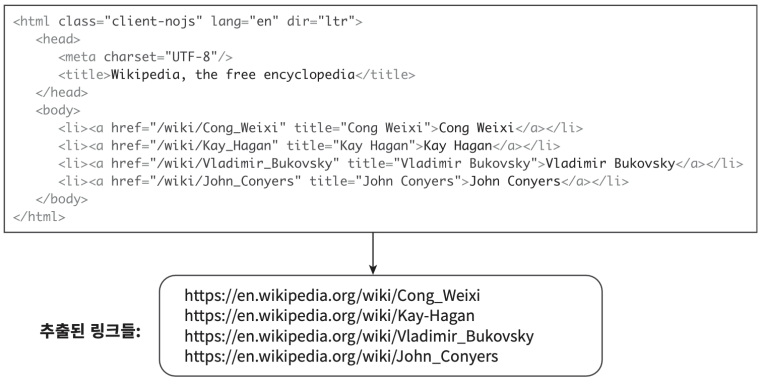

그림 9-3

URL 필터

- 특정 콘텐츠 타입이나 파일 확장자를 갖는 URL 접속 시 오류가 발생하는 URL, 접근 제외 목록(deny list)에 포함된 URL 등을 크롤링 대상에서 배제하는 역할을 함

이미 방문한 URL?

- 방문한 URL이나 미수집 URL 저장소에 보관된 URL을 추적할 수 있도록 하는 자료 구조 사용
- 이 자료 구조로는 블룸 필터나 해시 테이블이 널리 사용된다.

URL 저장소

- 이미 방문한 URL을 보관하는 저장소

> 위 컴포넌트들이 어떻게 상호 연동하는 과정을 작업 흐름(workflow) 관점에서 살펴본다.

웹 크롤러 작업 흐름

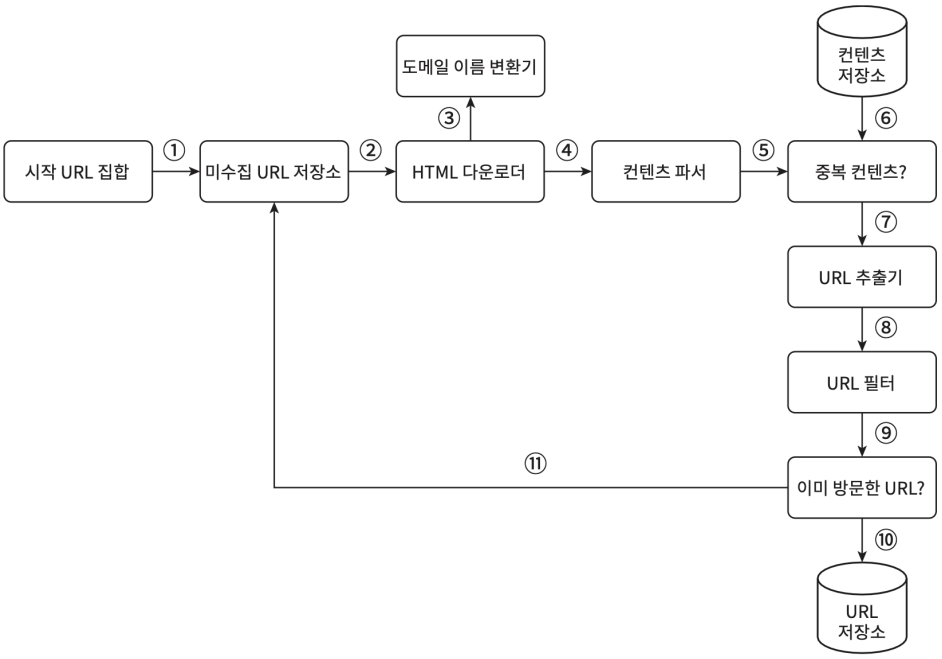

그림 9-4

1. 시작 URL들을 미수집 URL 저장소에 저장
2. HTML 다운로더는 미수집 URL 저장소에서 URL 목록을 가져온다.
3. HTML 다운로더는 도메인 이름 변환기를 사용하여 URL의 IP 주소를 알아내고, 해당 IP 주소로 접속하여 웹 페이지를 다운받는다.
4. 콘텐츠 파서는 다운된 HTML 페이지를 파싱하여 올바른 형식을 갖춘 페이지인지 검증한다.
5. 콘텐츠 파싱과 검증이 끝나면 중복 컨텐츠인지 확인하는 절차를 캐시한다.
6. 중복 컨텐츠인지 확인하기 위해서, 해당 페이지가 이미 저장소에 있는지 본다.
    - 이미 저장소에 있는 콘텐츠인 경우 처리하지 않고 버림
    - 저장소에 없는 컨텐츠인 경우에는 저장소에 저장한 뒤 URL 추출기로 전달
7. URL 추추기는 해당 HTML 페이지에서 링크를 골라낸다.
8. 골라낸 링크를 URL 필터로 전달한다.
9. 필터링이 끝나고 남은 URL만 중복 URL 판별 단계로 전달된다.
10. 이치 처리한 URL인지 확인하기 위해, URL 저장소에 보관된 URL인지 살핀다.
    이미 저장소에 있는 URL은 버린다.
11. 저장소에 없는 URL은 URL 저장소에 저장할 뿐 아니라 미수집 URL 저장소에도 전달한다.

3단계 : 상세 설계

DFS vs BFS

- 웹은 유향 그래프(directed graph)나 같다. 페이지는 노드이고, 하이퍼링크(URL)는 엣지(edge)라고 생각
- 크롤링 프로세스는 이 유향 그래프를 엣지를 따라 탐색하는 과정
  - 그래프를 엣지를 따라 탐색하는 과정은 DFS와 BFS 그래프 탐색 알고리즘이 있다.
  - DFS는 좋은 선택이 아닐 가능성이 높다. → 그래프의 크기가 클 경우 깊이를 가늠하기 어려움
- 따라서 웹 크롤러는 **보통 BFS를 사용**한다.
  - BFS는 FIFO 큐를 사용하는 알고리즘이다. 한쪽으로는 탐색할 URL 다른 쪽에선 꺼내기만 함
  - 단점으로는 한 페이지에서 나오는 링크의 상당수는 같은 서버로 되돌아가는 단점이 있다.
    - ex) **위키피디아**를 보면 **페이지에서 추출한 모든 링크는 내부 링크** 즉, **동일한 위키피디아 서버의 페이지를 참조하는 링크**다. **크롤러는 같은 호스트에 속한 많은 링크를 다운받느라** 바빠지게 되는데, 이때 링크들을 병렬로 처리하게 된다면 위키피디아 서버는 많은 요청으로 과부하게 걸리게된다.
      이런 크롤러는 보통 예의 없는(impolite) 크롤러로 간주된다.

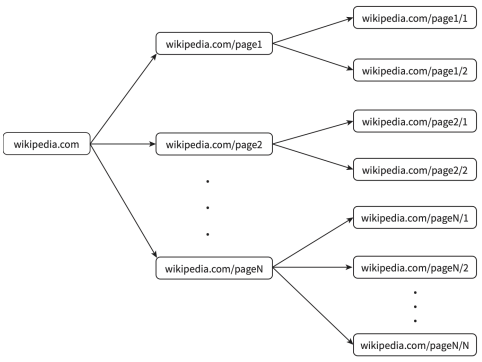

그림 9-5

- 표준적 BFS 알고리즘은 URL 간에 우선순위를 두지 않는다.
  - 모든 웹 페이지는 **품질**과 **수준**이 다르다. 그러니 **페이지 순위(page rank)**, **사용자 트래픽의 양**,

    업데이트 빈도 등 **여러 가지 척도로 처리 우선순위를 구별**하는 것이 좋다.

미수집 URL 저장소

- **URL 저장소**를 **잘 구현**하면 예의를 갖추고, URL 사이의 우선순위를 구별하는 크롤러를 구현할 수 있다.
    - 미수집 URL 저장소의 구현 방법에 대한 좋은 논문 [5], [9]을 참고

좋은 두 논문 요약

▼ **예의**

- 웹 크롤러는 수집 대상 서버로 짧은 시간 안에 너무 많은 요청을 보내는 것을 삼가야 한다.
    - 많은 요청을 보내는 것은 무례한 일이고, DoS 공격으로 간주되기도 함
- 예의 바른 크롤러를 만드는 데 있어 지켜야할 한 가지 원칙은, 동일 웹 사이트에 대해서는 한 번에 한 페이지만 요청
    - 같은 웹 사이트의 페이지를 다운받는 테스크는 웹사이트의 호스트명(hostname)과 다룬로드를 수행하는 작업 스레드(worker thread) 사이의 관계를 유지하면 된다.
    - 즉, 각 다운로드는 별도 FIFO 큐를 가지고 있어서, 해당 큐에서 꺼낸 URL만 다운로드 한다.

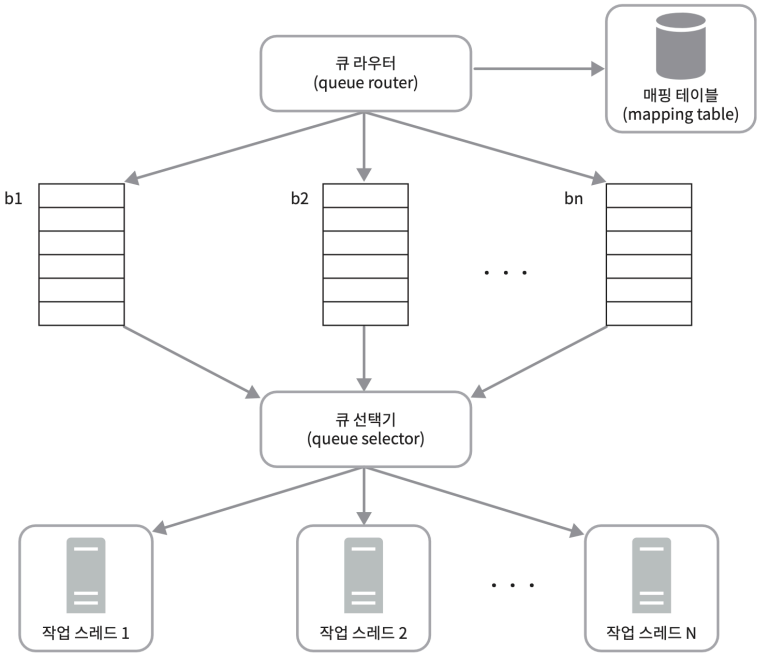

그림 9-6

- **큐 라우터(queue router)**
  - 같은 호스트에 속한 URL은 언제나 같은 큐(b1, b2, ..., bn)로 가도록 보장하는 역할
- **매핑 테이블(mapping table)**

호스트	큐
wikipedia.com	b1
apple.com	b2
...	...
nike.com	bn

표 9-1

  - 위 그림과 같이 호스트 이름과 큐 사이의 관계를 보관하는 테이블
- **FIFO 큐(b1부터 bn까지)**
  - 같은 호스트에 속한 URL은 언제나 같은 큐에 보관
- **큐 선택기(queue selector)**

- 큐 선택기는 큐들을 순회하면서 큐에서 URL을 꺼내서 해당 큐에서 나온 URL을 다운로드하도록 지정된 작업 스레드에 전달하는 역할

- **작업 스레드(worker thread)**
  - 작업 스레드는 전달된 URL을 다운로드하는 작업을 수행
  - 전달된 URL은 순차적으로 처리될 것이며, 작업들 사이에는 일정한 지연시간을 둘 수 있음

▼ **우선순위**

- 다양한 방식이 있겠지만 여기서는 **순위결정장치(prioritizer)**를 살펴본다. ([5]와 [10]를 참고)

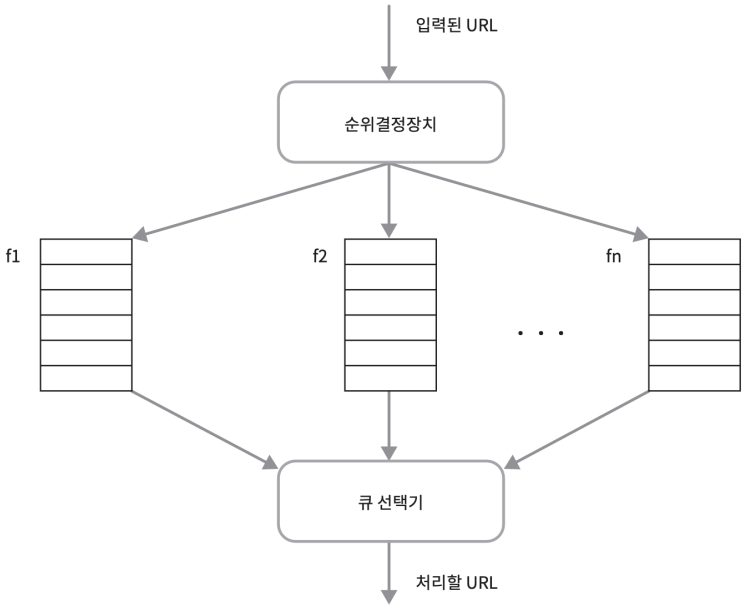

그림 9-7

- **순위결정장치(prioritizer)**
  - URL을 입력으로 받아 우선순위를 계산한다.
- **큐(f1, ..., fn)**
  - 우선순위별로 큐가 하나씩 할당된다.
  - 우선순위가 높으면 선택될 확률도 올라간다.
- **큐 선택기**
  - 임의 큐에서 처리할 URL을 꺼내는 역할을 담당한다.
  - 순위가 높은 큐에서 더 자주 꺼내도록 프로그램되어 있다.

- 예의와 우선순위를 반영한 전체 설계

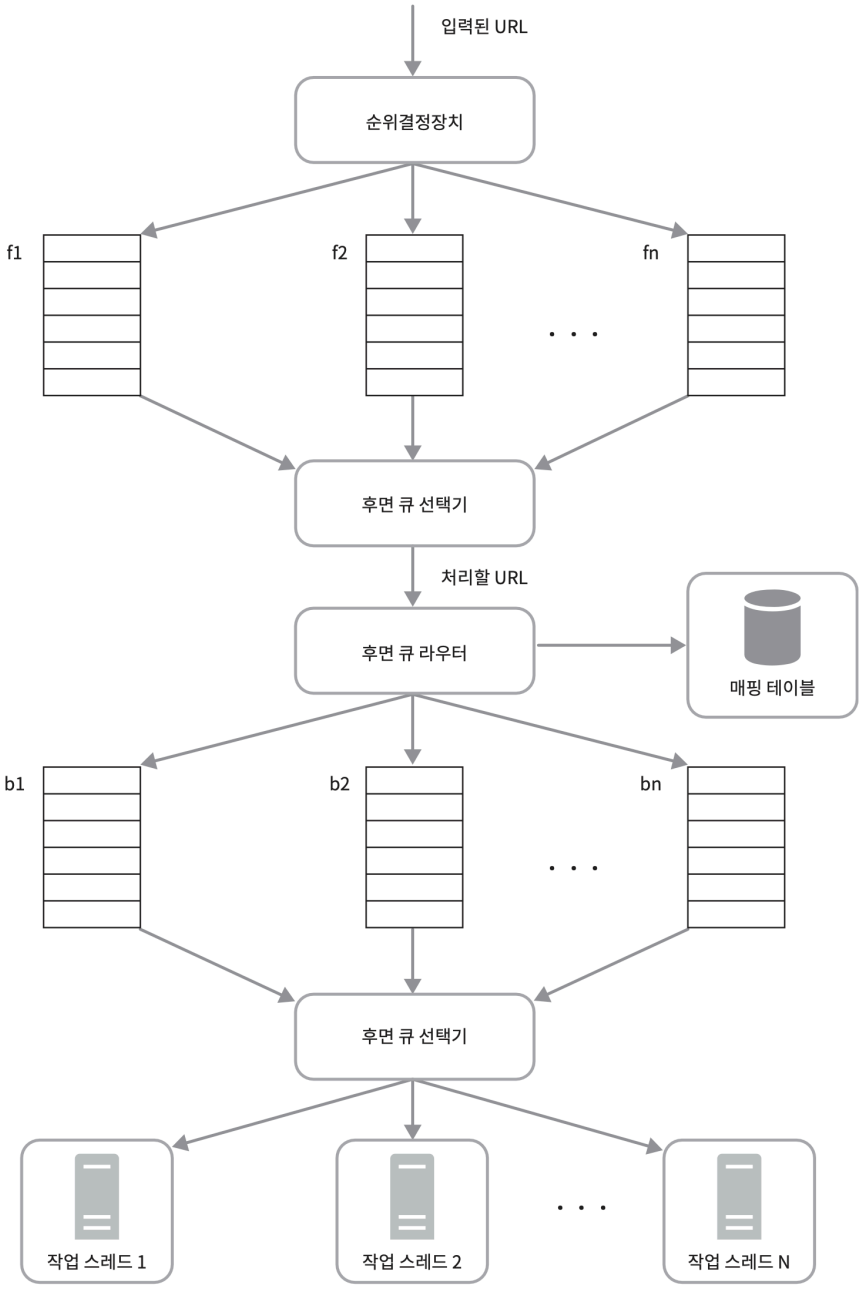

그림 9-8

  - 전면 큐(front queue) 우선순위 결정 과정을 처리한다.
  - 후면 큐(back queue) 크롤러가 예의 바르게 동작하도록 보증한다.

▼ 신선도

- 데이터의 신선함을 유지하기 위해서는 이미 다운로드 했어도 주기적으로 재수집할 필요가 있음

- 모든 URL을 재수집하는 것은 낭비임 (자주 변하지 않는 웹 사이트들도 존재)
- **신선도 최적화를 위한 전략**
  - 웹 페이지의 변경 이력(update history) 활용
  - 우선순위를 활용하여, 중요한 페이지는 좀 더 자주 재수집

▼ **미수집 URL 저장소를 위한 지속성 저장장치**

- **검색 엔진**을 위한 **크롤러**의 경우 처리해야 하는 **URL의 수는 수억 개** 이상이다.
  이 정보를 메모리에 보관하는 것은 안정성이나 규모 확작성에서 좋지 않다.
  그렇다고 전부 디스크에 저장하는 것도 성능상 좋지 않다.
- 따라서 이 책의 설계에서는 모두 사용하는 **혼합 접근법(hybrid approach)**을 선택
  - **대부분의 URL은 디스크**에 두지만 **IO 비용을 줄이기 위해 메모리 버퍼에 큐**를 두는 것
    버퍼에 있는 데이터는 주기적으로 디스크에 기록
  - 캐시를 쓰는 상황과 비슷?

HTML 다운로더

- HTTP 프로토콜을 통해 웹 페이지를 내려받는다.
- 이를 위해 `robot.txt 파일`과 `성능 최적화`가 중요하다.

▼ **robot.txt 파일**

- **로봇 제외 프로토콜(Robot Exclusion Protocol)**이라고 부르기도 함
- 크롤러가 수집해도 되는 페이지의 목록이 들어 있음 (크롤러는 이를 잘 확인해야 함)
- robots.txt 파일이 여러번 다운로드 되는 것을 피하기 위해 주기적으로 캐시에 보관
- aws 예시 - https://www.amazon.com/robots.txt

```
User-agent: *
Disallow: /exec/obidos/account-access-login
Disallow: /exec/obidos/change-style
                .
                .
                .
```

▼ **성능 최적화**

1. 분산 크롤링

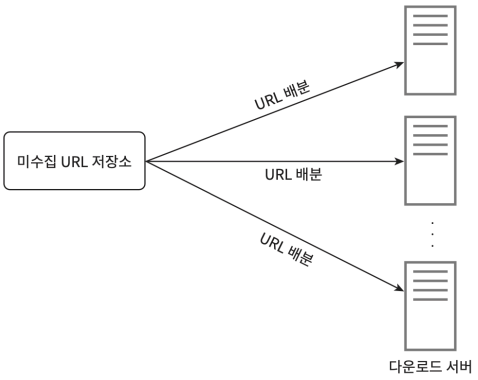

그림 9-9

- 크롤링 작업을 여러 서버에 분산하는 방법
- 각 서버는 여러 스레드를 돌려 다운로드 작업 처리
- 이를 위해 URL 공간은 작은 단위로 반할 (각 서버는 그중 일부 다운로드 담당)

2. 도메인 이름 변환 결과 캐시
   - **도메인 이름 변환기(DNS Resolver)는 동기적 특성**이 띄어 보통 10ms에서 200ms가 소요
     - DNS 요청을 보내고 결과를 받는 작업이 동기적일 수 밖에 없음
   - DNS 조회 결과로 얻어진 도메인 이름과 IP 주소 사이의 관계를 캐시에 보관.
     - 크론 잡(cron job) 등을 돌려 주기적으로 갱신사도록 해놓자.
3. 지역성
   - 크롤링할 웹 서버의 지리적으로 가깝게 하기 위해 크롤링 서버를 지역별로 분산
   - 지역성을 활용하는 전략은 크롤 서버, 캐시, 큐, 저장소 등 대부분의 컴포넌트에 적용 가능
4. 짧은 타임아웃
   - 최대 얼마나 기다릴지 미리 정해둬야 한다.
   - 이 시간 동안 서버가 응답하지 않으면 해당 페이지에 다운로드를 중단하고 다음 페이지로 이동

안정성 확보 전략

- **안정 해시(consistent hashing)**
  - 다운로더 서버들에 부하를 분산할 때 적용 가능한 기술
  - 이 기술을 이용하면 다운로더 서버를 쉽게 추가하고 삭제할 수 있음
- **크롤링 상태 및 수집 데이터 저장**
  - 장애가 발생한 경우에도 쉽게 복구할 수 있도록 크롤링 상태와 수집된 데이터를 지속적 저장장치에 기록해 두는 것이 좋음
  - 저장된 데이터를 로딩하고 나면 중단되었던 크롤링을 쉽게 재시작할 수 있음
- **예외 처리(exception handling)**
  - 대규모 시스템에서 에러(error)는 불가피할 뿐 아니라 흔하게 벌어짐
  - 예외가 발생해도 전체 시스템이 중단되는 일 없이 그 작업을 이어나갈 수 있어야 함
- **데이터 검증(data validation)**
  - 시스템 오류를 방지하기 위한 중요 수단 가운데 하나

확장성 확보 전략

- 새루온 형태의 컨텐츠(웹, 이미지, PDF 등..)를 쉽게 지원할 수 있도록 신경 쓰는 것이 좋음
- 책의 예제의 경우 새로운 모듈을 끼워 넣으므로 새로운 형태의 콘텐츠를 지원할 수 있도록 설계

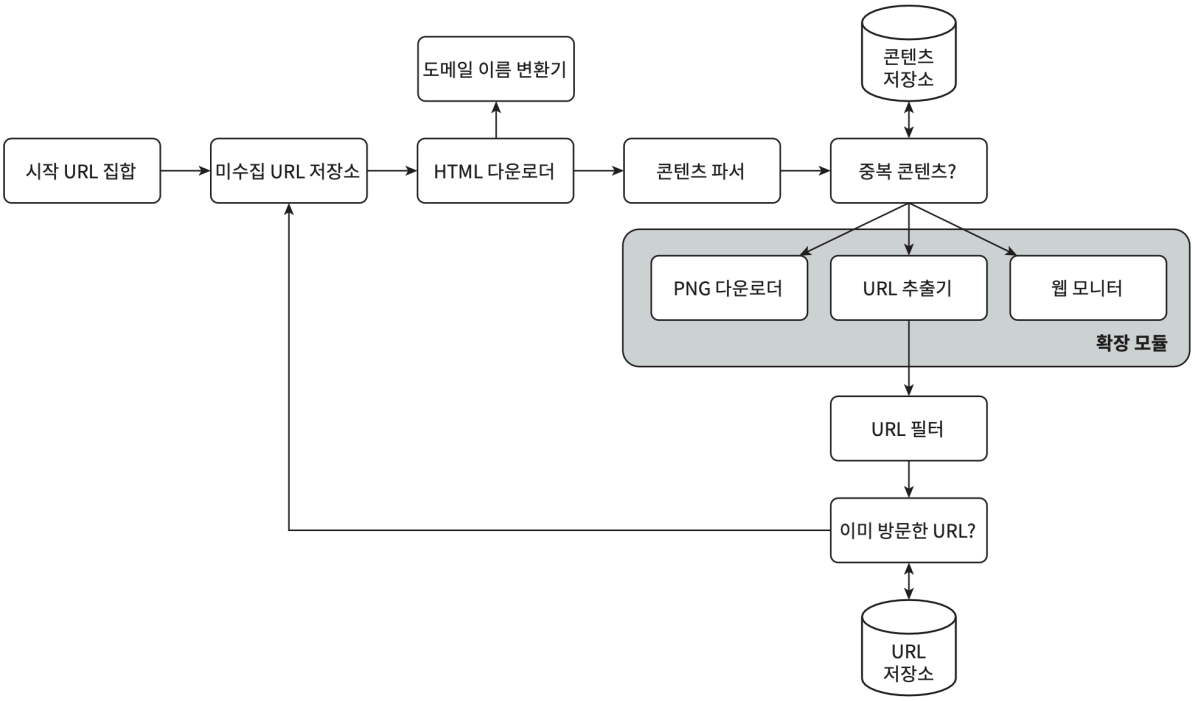

그림 9-10

- PNG 다운로더는 PNG 파일을 다운로드하는 플러그인(plug-in) 모듈
- 웹 모니터는(web monitor)는 웹을 모니터링하여 저작권이나 상표권이 침해되는 일을 막는 모듈

문제 있는 콘텐츠 감지 및 회피 전략

1. **중복 컨텐츠**
    - 웹 콘텐츠의 30% 가량은 중복.
    - **해시**나 **체크섬(checksum)**을 사용해 중복 컨텐츠를 보다 쉽게 탐지 가능
        - 이를 어떻게 구현할 수 있을까?, `URL 자체`로?, URL 안에 `포함된 내용`이 중복이라면?
2. **거미 덫(spider trap)**
    - 크롤러를 무한 루프에 빠지게 설계된 페이지.

    > ex) 다음과 같이 무한히 깊은 디렉터리 구조를 포함하는 링크가 있을 경우
    >
    > spidertrap.com/foo/bar/foo/bar/foo/...

- 이런 덫은 **URL의 최대 길이를 제한하여 회피**할 수 있음
  하지만 가능한 모든 종류의 덫을 피할 수 있는
  만능 해결책은 없음
- 이런 덫이 설치된 웹 사이트를 알아내는 것은 어렵지 않아, 수작업으로 처리해도 좋음

3. **데이터 노이즈**
   - 광고나 스크립트 코드, 스팸 URL 같은 가치가 없는 컨텐츠만 있을 경우
   - 2번처럼 회피해야 할 듯?

4단계 : 마무리

추가로 논의할 내용

- **서버 측 랜더링(server-side rendering)**
  - 많은 웹 사이트가 JS, AJAX 등의 기술을 사용해 링크를 즉석해서 만들어 낸다.
  - 따라서 웹 페이즈를 있는 그대로 다운받아서 파싱해보면 동적으로 생성되는 링크는 발견할 수 없다.
  - 이때 페이지를 파싱하기 전 **서버 측 랜더링(동적 랜더링이라고도 함)**을 적용하면 해결할 수 있음
- **원치 않는 페이지 필터링**
  - 크롤링에 소요되는 자원은 유한하기 때문에 스팸 방지(anti-spam) 컴포넌트를 두어 품질이 조악하거나 스팸성인 페이지를 걸러내면 좋다. ([13], [14] 참고)
- **데이터베이스 다중화(replication) 및 샤딩(sharding)**
  - 다중화나 샤딩 같은 기법을 적용하면 데이터 계층(data layer)의 가용성, 규모 확장성, 안정이 향상된다.
- **수평적 규모 확장성(horizontal scalability)**
  - 다운로드를 실행할 서버가 수백, 수천 대가 될 수 있으므로 **무상태(stateless) 서버**로 만드는 것이 좋다.
- **가용성, 일관성, 안정성**
  - 성공적인 대형 시스템을 만들기 위해 필수적으로 고려해야 하는 사항들이다.

- **데이터 분석 솔루션(analytics)**
  - 데이터를 수집하고 분석하는 것은 어느 시스템이나 중요.

▼ Chapter 9: DESIGN A WEB CRAWLER

1. US Library of Congress
2. EU Web Archive
3. Digimarc
4. Mercator: A Scalable, Extensible Web Crawler
5. Web Crawling By Christopher Olston and Marc Najork
6. 29% Of Sites Face Duplicate Content Issues
7. Michael O. Rabin (1981). "Fingerprinting by Random Polynomials"
8. Bloom (1970) Space/Time Trade-offs in Hash Coding with Allowable Errors
9. Web Crawling - Introduction to Information Retrieval Donald J. Patterson
10. The PageRank Citation Ranking: Bringing Order to the Web
11. Bloom (1970) Space/Time Trade-offs in Hash Coding with Allowable Errors
12. Google Dynamic Rendering
13. Tracking Web Spam with Hidden Style Similarity - Tanguy Urvoy, Thomas Lavergne, Pascal Filoche
14. IRLbot: Scaling to 6 Billion Pages and Beyond - Hsin-Tsang Lee, Derek Leonard, Xiaoming Wang, and Dmitri Loguinov