9️⃣

9장. 웹 크롤러 설계

> 웹에 새로 올라오거나 갱신된 콘텐츠를 찾아내는 것이 주된 목적

크롤러 활용 예시

- 검색 엔진 인덱싱 : 크롤러의 가장 보편적인 용례. 검색 엔진을 위한 로컬 인덱스를 만든다.
- 웹 아카이빙 : 나중에 사용할 목적으로 장기보관하기 위해 웹에서 정보를 모으는 절차를 말한다. 많은 국립 도서관이 크롤러를 돌려 웹사이트를 아카이빙한다.
- 웹 마이닝: 웹 마이닝을 통해 인터넷에서 유용한 지식을 도출해 낼 수 있다.
- 웹 모니터링: 크롤러를 사용하면 인터넷에서 저작권이나 상표권이 침해되는 사례를 모니터링할 수 있다.

웹크롤러의 복잡도는 처리해야 하는 **데이터의 규모** 에 따라 달라진다.

몇 시간이면 끝낼 수 있는 작은 학급 프로젝트 수준일 수도 있고, 별도의 엔지니어링 팀을 꾸려서 지속적으로 관리하고 개선해야 하는 초대형 프로젝트가 될 수 있다.

→ 따라서, 크롤러 설계의 가장 우선은 **데이터의 규모와 기능**을 정의하는 것이다.

1단계. 문제 이해 및 설계 범위 확정

웹 크롤러의 기본 알고리즘

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

웹 크롤러가 가져야할 특성

- 규모 확장성: parallelism을 활용하면 보다 효과적으로 웹 크롤링을 할 수 있을 것이다.

- 안정성(robustness): 잘못 작성된 HTML, 아무 반응이 없는 서버, 장애, 악성코드가 붙어 있는 링크 등 비정상적 입력이나 환경에 대응할 수 있어야한다.
- 예절(politeness): 수집 대상 웹 사이트에 짧은 시간 동안 너무 많은 요청을 보내서는 안된다.
- 확장성(extensibility): 새로운 형태의 콘텐츠를 지원하기가 쉬워야한다. ex) 갑자기 이미지 파일도 크롤링 하고 싶어짐 👉🏻 조금만 확정해도 될 수 있는 유연성 필요

개략적 규모 측정

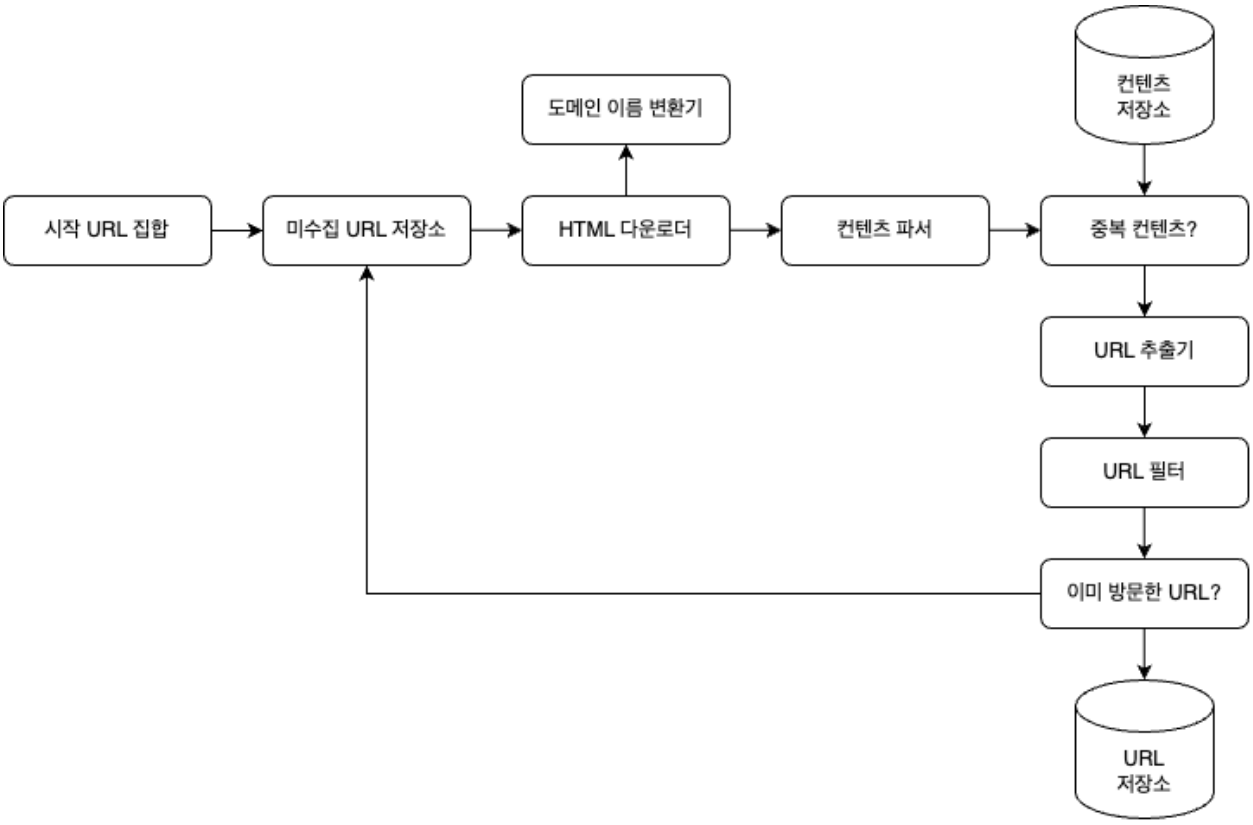
조회 연산량

- 매달 10억개의 웹 페이지를 다운로드한다.
- QPS = 10억/30일/24시간/3600초 = 대략 400페이지/초
- 최대(peak) QPS = 2 * QPS = 800

저장 용량

- 웹 페이지의 크기 평균은 500k라고 가정
- 10억 페이지 * 500k = 500TB/월
- 1개월치 데이터를 보관하는 데는 500TB, 5년간 보관한다고 가정하면 결국 500TB *12개월* 5년 = 30PB의 저장용량이 필요

2단계. 개략적 설계안 제시 및 동의 구하기



시작 URL 집합

> 웹 크롤러가 작동하는 출발점

- 단순 예시) 어떤 대학 웹사이트로부터 찾아 나갈 수 있는 모든 웹페이지를 크롤링하는 가장 직관적인 방법
  - 해당 대학의 도메인 이름이 붙은 모든 페이지의 URL을 시작 URL로 쓴다.
- 확장) 전체 웹을 크롤링해야하는 경우 👉🏻 URL을 고르는 방법을 고안해야함
  - 가능한 한 많은 링크를 탐색할 수 있도록 하는 URL을 고르는 것이 바람직.
  - 전체 URL 공간을 작은 부분집합으로 나누는 전략을 쓴다.
  - 주제별로 다른 시작 URL을 사용

시작 URL을 무엇으로 쓸 것인지는 정답 X, 의도만 잘 설명하자

미수집 URL 저장소

대부분의 현대 웹 크롤러는 크롤링 상태를

1. 다운로드 **할** URL
2. 다운로드 **된** URL

로 나눠 관리한다. 그리고 첫번째 상태를 저장하는 컴포넌트를 미수집 URL 저장소 (URL frontier)라고 부른다. FIFO 큐라고 생각하면 된다.

HTML 다운로더

인터넷에서 웹페이지를 다운로드하는 컴포넌트, 다운하려는 URL은 URL 프론티어에서 받아온다.

도메인 이름 변환기

> URL → IP

HTML 다운로더가 이를 위해 도메인 이름 변환기를 이용한다.

콘텐츠 파서

웹페이지를 다운로드하면 파싱과 검증 절차를 거쳐야한다.

→ 크롤링 서버 안에 콘텐츠 파서를 구현하면 크롤링 과정이 느려질 수 있다.

중복 콘텐츠인가?

웹에 공개된 연구결과에 따르면, 29% 가량의 웹 페이지 컨텐츠는 중복이다.

→ 따라서 같은 컨텐츠를 여러번 저장하게 될 수 있다. 본 설계안의 경우, 이 문제를 해결하기 위한 **자료구조를 도입**하여 데이터 중복을 줄이고 데이터 처리에 소요되는 시간을 줄인다.

자료구조 도입 -> 이미 시스템에 저장된 컨텐츠임을 알아내기 쉽게 함

두 HTML 파일을 효율적으로 비교하기 위해 **해싱**을 사용할 수 있음.

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 고려
- 본 설계안의 경우에는 디스크와 메모리를 동시에 사용하는 저장소를 택할 것이다.
    - 데이터 양이 너무 많으므로 대부분의 컨텐츠는 디스크에 저장한다.
    - 인기있는 컨텐츠는 메모리에 두어 접근 지연시간을 줄일 것이다.

URL 추출기

저장한 HTML 문서들을 파싱해 링크들을 골라내는 역할.

상대경로는 전부 절대경로로 변환해준다.

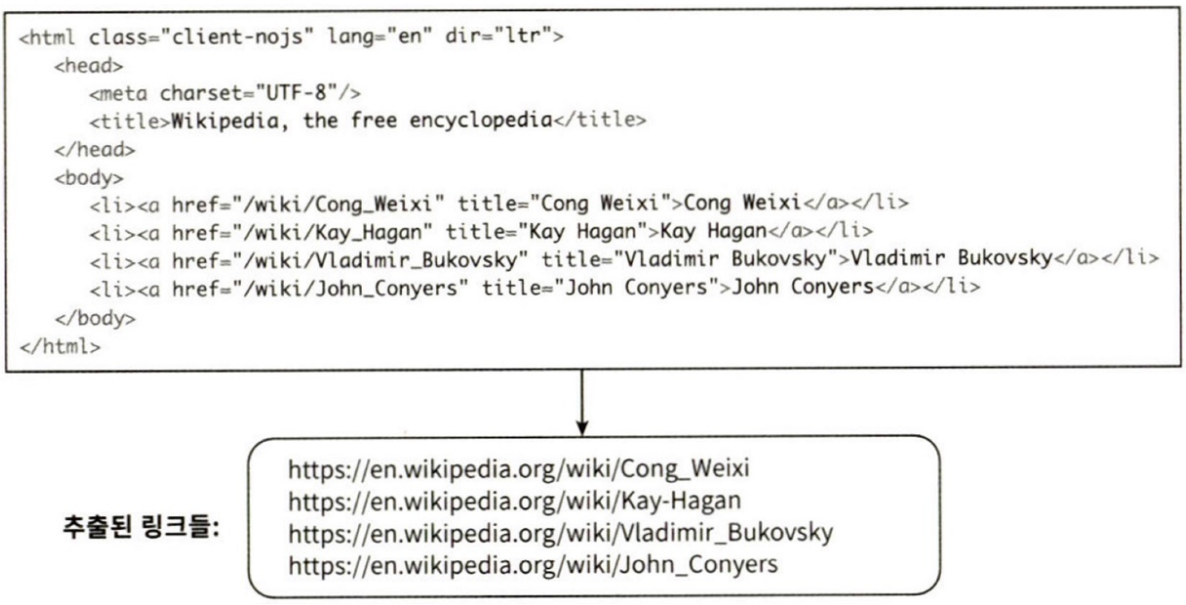


그림 9-3

URL 필터

URL 필터는, URL 저장소에 저장하기 전

- 특정한 컨텐츠 타입이나 파일 확장자를 갖는 URL
- 접속 시 오류가 발생하는 URL
- 접근 제외 목록(deny list) 에 포함된 URL

을 필터링한다.

이미 방문한 URL?

- 이미 방문한 URL 저장소
- 미수집 URL 저장소

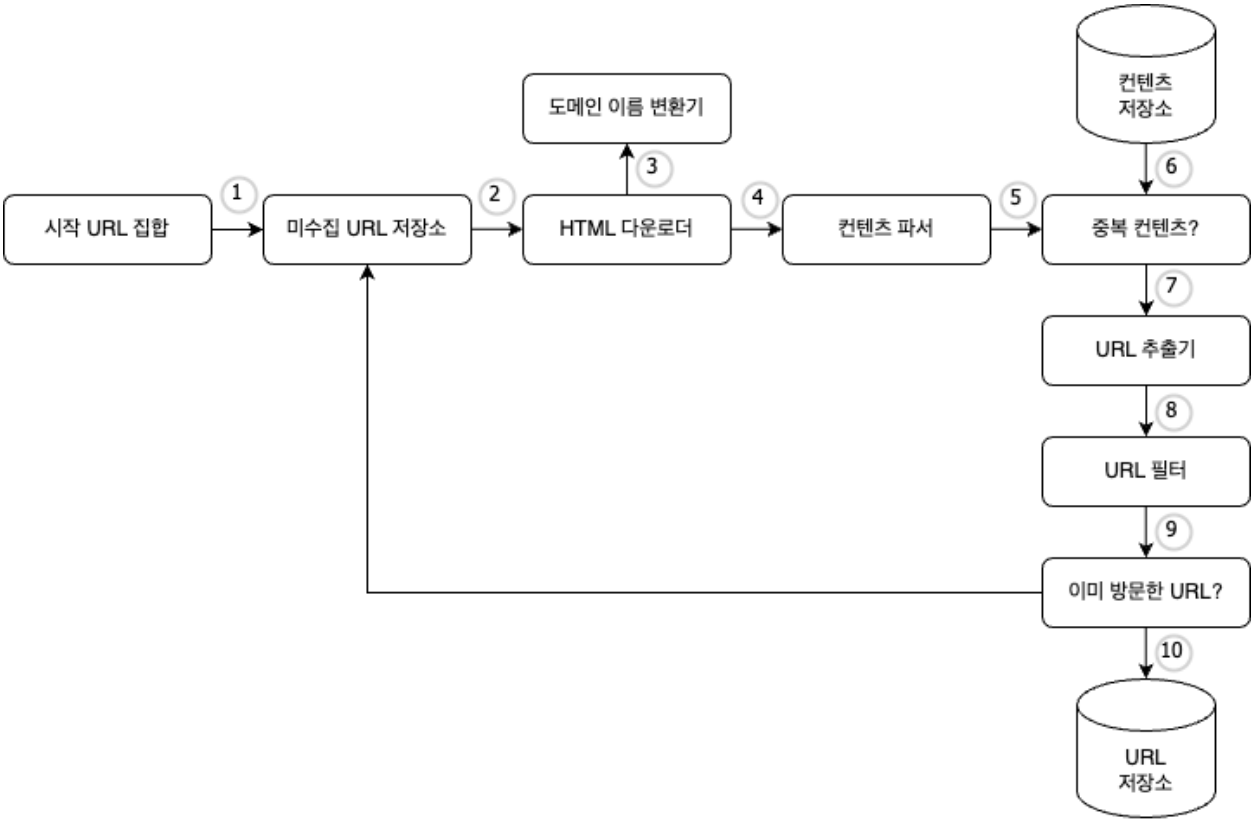
등을 구현해 보관된 URL을 추적할 수 있게 하는 자료구조를 만든다

해당 자료 구조로는 **블룸 필터**나 **해시 테이블**이 널리 쓰인다.

URL 저장소

> 이미 방문한 URL을 보관하는 저장소

전체 작업 흐름



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 웹페이지를 다운받는다.
4. 컨텐츠 파서는 다운된 HTML 페이지를 파싱하여 올바른 형식을 갖춘 페이지인지 검증한다.
5. 컨텐츠 파싱과 검증이 끝나면 중복 컨텐츠인지 확인하는 절차를 개시한다.
6. 중복 컨텐츠인지 확인하기 위해서, 해당 페이지가 이미 저장소에 있는지 본다.
    - 이미 저장소에 있는 컨텐츠인 경우에는 처리하지 않고 버린다
    - 저장소에 없는 컨텐츠인 경우에는 저장소에 저장한 뒤 URL 추출기로 전달한다.
7. URL 추출기는 해당 HTMI 페이지에서 링크를 골라낸다.
8. 골라낸 링크를 URL 필터로 전달한다.
9. 필터링이 끝나고 남은 URL만 중복 URL 판별 단계로 전달한다.
10. 이미 처리한 URL인지 확인하기 위하여, URL 저장소에 보관된 URL인지 살핀다. 이미 있으면 버림
11. 저장소에 없는 URL은 URL 저장소에 저장할 뿐 아니라 미수집 URL 저장소에도 전달한다.

3단계. 상세 설계

가장 중요한 컴포넌트 & 구현 기술을 상세하게 보자

- DFS vs BFS
- 미수집 URL 저장소
- HTML 다운로더
- 안정성 확보 전략
- 확장성 확보 전략
- 문제 있는 컨텐츠 감지 및 회피 전략

DFS? BFS?

웹은 유향 그래프와 같다.

페이지 == 노드, 하이퍼링크 == 엣지

→ 크롤링 == 그래프를 탐색하는 과정

DFS, BFS는 바로 이 그래프 탐색에 널리 사용되는 두가지 알고리즘이다. 하지만 DFS는 좋은 선택이 아닐 가능성이 높다. 그래프 크기가 클 경우 어느정도로 깊숙이 가게 될지 가늠하기 어려워서다.

→ 따라서 웹 크롤러는 보통 **BFS**를 사용한다. 큐를 사용하는 알고리즘으로, 한쪽으로는 탐색할 URL을 집어넣고, 다른 한 쪽으로는 꺼내기만한다.

그런데 이 구현법에는 두 가지 문제점이 있다.

1. 한 페이지에서 나오는 링크의 상당수는 같은 서버로 되돌아감. => 예절(politeness) 위배
2. 우선순위가 없다. 페이지 순위, 사용자 트래픽의 양, 업데이트 빈도 등 여러가지 척도에 비추어 구현하려면 표준적 알고리즘만 사용해서는 안된다.

→ 위의 문제들을 미수집 URL 저장소로 쉽게 해결 가능

미수집 URL 저장소 & 구현 방법

> 다운로드할 URL을 보관하는 장소

예의

동일 웹사이트에 대해서는 한 번에 한 페이지만 요청한다.

같은 페이지를 다운받는 태스크는 시간차를 두고 실행한다.

구현 방법 👉 웹사이트의 호스트명과 다운로드를 수행하는 작업 스레드 사이의 관계를 유지하면 된다. 각 다운로드 스레드는 별도 FIFO 큐를 가지고 있어서 해당 큐에서 꺼낸 URL 만 다운로드 한다.

→ 호스트별로 들어가는 큐를 지정해주면 됨. 같은 큐에서 나온 것은 동시에 접속하지 않는다.

우선순위

유용성에 따라 URL의 우선순위를 나눌 때는 **페이지 랭크, 트래픽 양, 갱신 빈도** 등 다양한 척도를 사용할 수 있을 것이다.

순위 결정 장치(prioritizer) ⇒ URL 우선순위를 정하는 컴포넌트다.

큐에 URL을 저장하기 전에 순위 결정 장치를 거치도록 설계를 변경한다.

1. 순위 결정 장치: URL 을 입력으로 받아 우선순위를 계산.
2. 우선순위 별로 큐가 하나씩 할당된다. 우선순위가 높으면 **선택될 확률**도 올라간다.
3. 큐 선택기: 임의 큐에서 처리할 URL을 꺼낸다. 순위가 높은 큐에서 더 자주 꺼내도록 프로그램 되어있다.

위의 두 기능을 반영한 전체 설계는 다음과 같다.

- 전면 큐 (front queue): 우선순위 결정 과정을 처리한다.
- 후면 큐 (back queue): 크롤러의 politeness를 보장한다.

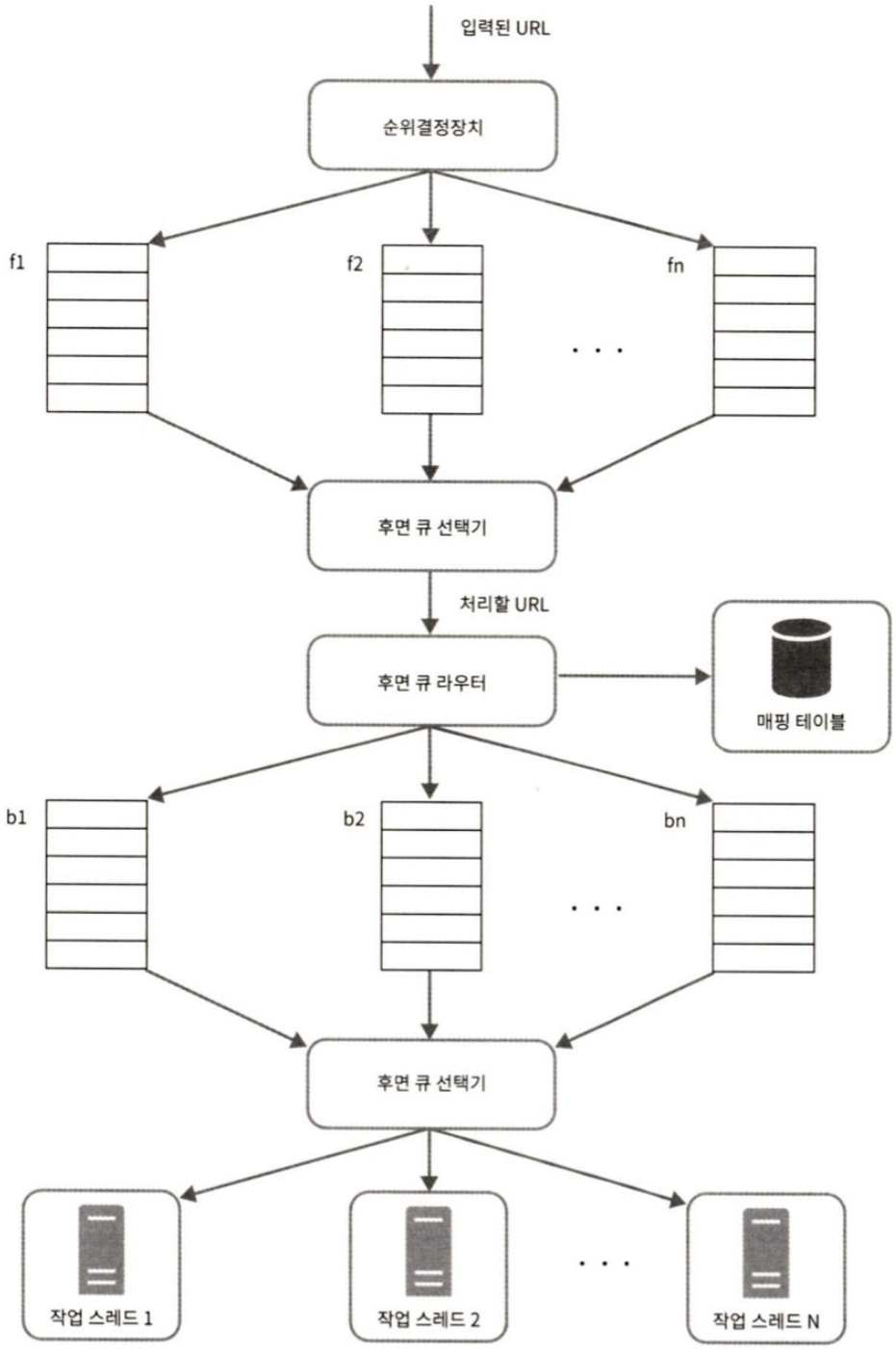


그림 9-8

신선도

웹페이지는 수시로 추가되고, 삭제되고, 변경되므로 데이터의 신선함(freshness)를 유지하기 위해서 이미 다운받은 페이지라고 해도 재수집할 필요가 있다.

이 작업을 최적화하기 위한 전략으로는 다음과 같은 것들이 있다.

- 웹 페이지의 변경 이력 활용
- 우선순위가 높은 페이지는 더 자주 재수집

미수집 URL 저장소를 위한 지속성 저장장치

검색 엔진을 위한 크롤러 => 수억개에 달하는 URL을 처리

그 URL들을 어디에 저장해야할까?

- 메모리?
  - ❌: 안정성, 규모 확장성이 떨어짐
- 디스크?
  - ❌: 느려서 쉽게 성능 병목지점이 됨
- 절충안 : 대부분의 URL은 디스크에 두지만 **메모리 버퍼에 `큐`를 둔다.(IO 비용 줄이기 위해)**

  버퍼에 있는 데이터는 주기적으로 디스크에 기록.

HTML 다운로더

HTML 다운로더는 HTTP 프로토콜을 통해 웹페이지를 내려 받음.

성능 최적화

1. 분산 크롤링

   성능을 높이기 위해 크롤링 작업을 여러 서버에 분산.

2. 도메인 이름 변환 결과 캐시

   DNS resolver는 크롤러 성능의 병목 중 하나인데, 이는 DNS 요청을 보내고 결과를 받는 작업의 동기적 특성 때문이다.

   스레드는 DNS 요청의 결과를 받기 전까지는 다음 작업을 진행할 수 없다. 크롤러 스레드 가운데 어느 하나라도 DNS resolver에 요청을 보내면, 이 요청이 완료될 때까지 다른 스레드의 요청이 모두 block된다.

   따라서 DNS 조회 결과로 얻어진 도메인 이름과 그에 상응하는 IP 주소를 캐시에 보관, 주기적으로 갱신하도록 구현하여 성능을 높일 수 있다.

3. 지역성

크롤링 작업을 수행하는 서버를 지역별로 분산하는 방법

크롤링 서버가 크롤링 대상 서버와 지역적으로 가까우면 페이지 다운로드 시간이 줄어들것이다. 지역성을 활용하는 전략은 크롤서버, 캐시, 큐, 저장소 등 대부분의 컴포넌트에 적용 가능하다.

4. 짧은 타임아웃

   어떤 웹서버는 응답이 느리거나 아예 응답하지 않는데, 이런 경우를 대비

   크롤러가 최대 얼마나 기다릴지를 미리 정해두는 것이다. 타임아웃의 경우 해당 페이지 다운로드를 중단한다.

안정성

1. 안정 해시(consistent hashing)

   다운로더 서버(분산 HTML 다운로더)들에 부하를 고르게 분산하기 위해 적용가능한 기술.

2. 크롤링 상태 및 수집 데이터 저장

   장애가 발생한 경우에도 쉽게 복구할 수 있도록 크롤링 상태와 수집된 데이터를 지속적 저장장치에 기록해두는 것이 바람직

3. 예외처리

   예외가 발생해도 전체 시스템이 중단되지 않도록 미리 처리

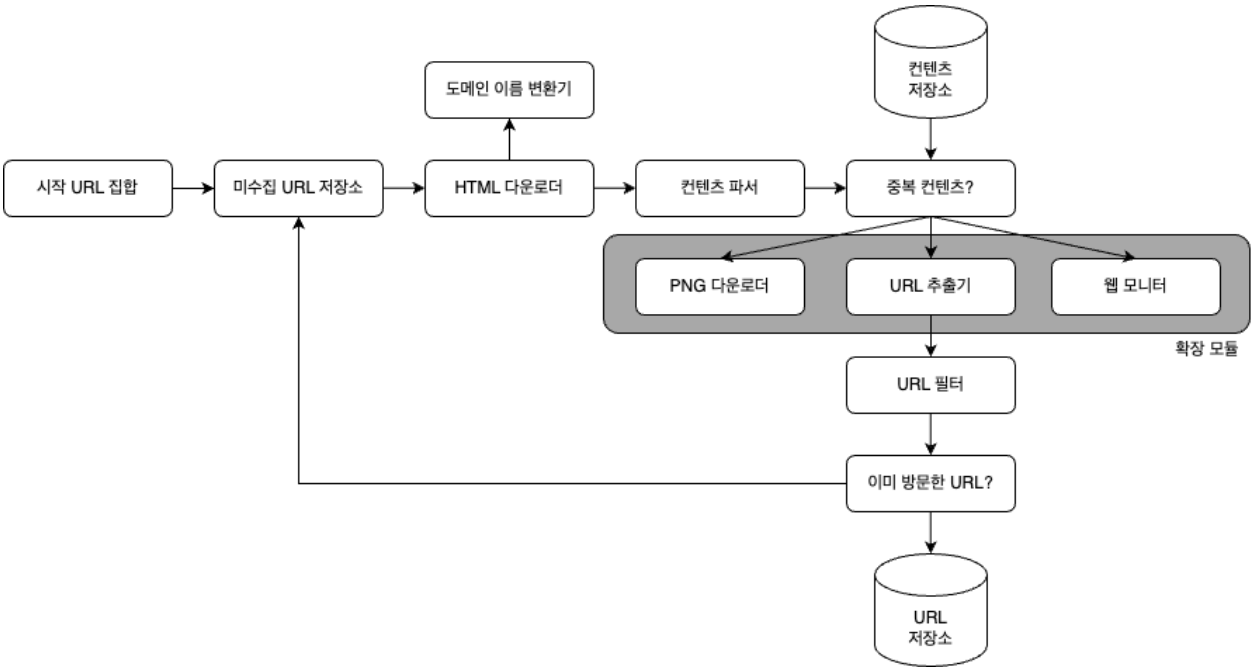
4. 데이터 검증

   시스템 오류를 방지하기 위한 중요 수단.

확장성

이런 시스템을 설계할 때는 새로운 형태의 컨텐츠를 쉽게 지원할 수 있도록 신경써야한다.

본 예제의 경우에는 새로운 모듈을 끼워 넣음으로써 새로운 형태의 컨텐츠를 지원할 수 있도록 설계하였다.



- PNG 다운로더는 PNG 파일을 다운로드 하는 plug-in 모듈
- 웹 모니터는 웹을 모니터링하여 저작권이나 상표권이 침해되는 일을 막는 모듈

문제 있는 콘텐츠 감지 및 회피

1. 중복 콘텐츠

   해시나 체크섬을 사용

2. 거미 덫

   크롤러를 무한 루프에 빠뜨리도록 설계한 웹 페이지

   ex) spidertrapexample.com/foo/bar/foo/bar/foo/bar/...

   - URL의 최대 길이를 제한 (모든 경우 다 피할 수는 없음)

   → 그래서 사람이 수작업으로 찾아낸 후 이런 사이트를 크롤러 탐색 대상에서 제외하거나 URL 필터 목록에 걸어둔다.

3. 데이터 노이즈

   광고, 스크립트 코드 등 → 이런 컨텐츠를 가능한 제외

4단계. 마무리

좋은 크롤러가 갖추어야하는 특성

- 규모 확장성(scalability)
- 예의(politeness)
- 확장성(extensibility)
- 안정성

규모 확장성이 뛰어난 웹 크롤러 설계 작업은 단순하지 않다. 웹이 워낙 방대한데다, 수없이 많은 덫이 도사리고 있기 때문

추가 논의 사항

- 서버 측 렌더링 (server-side rendering)

  많은 웹사이트가 자바스크립트, ajax 등의 기술을 사용해서 링크를 즉석에서 만들어낸다.

  그러니 웹 페이지를 그냥 있는 그대로 다운받아서 파싱해보면 그렇게 동적으로 생성되는 링크는 발견할 수 없을 것이다. 이 문제는 페이지를 파싱하기 전에 서버 측 렌더링을 적용하면 해결할 수 있다.

- 원치 않는 페이지 필터링

  스팸 방지 컴포넌트를 두어 품질이 조약하거나 스팸성인 페이지를 걸러내도록 하면 좋다.

- 데이터베이스 다중화 및 샤딩

  데이터 계층 가용성, 규모확장성, 안정성

- 수평적 규모 확장성

  대규모 분산 다운로드 서버로 확장하기 위해 무상태 서버로 만들기

- 가용성, 일관성, 안정성
- 데이터 분석 솔루션(analytics)

참고 링크

https://velog.io/@kyy00n/대규모-시스템-설계-기초-9장.-웹-크롤러-설계