9장. 웹 크롤러 설계

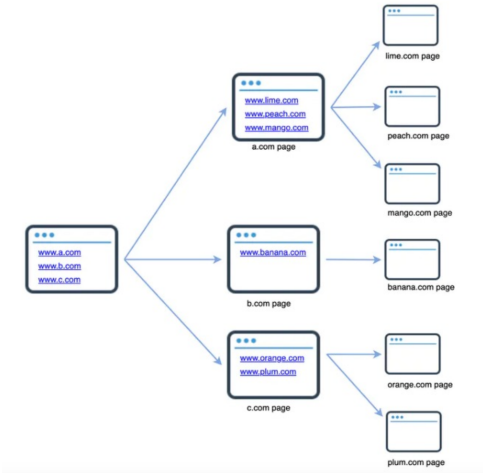
Created	@November 3, 2022 11:48 AM
Progress	In Progress

📃 **학습 TODO list**

- [ ]

`웹 크롤러(web crawler)`(로봇(robot), 스파이더(spider))

- 목적: 웹에 새로 올라오거나 갱신된 `콘텐츠`(웹 페이지, 이미지, 비디오, PDF 파일 등)를 찾아낸다.
- 몇 개 웹 페이지에서 시작해서 그 링크를 따라 나가면서 새로운 `콘텐츠`를 수집한다.



- `웹 크롤러`의 복잡도는 `웹 크롤러`가 처리해야 하는 데이터의 규모에 따라 달라진다.
  - 감당해야 하는 데이터의 규모와 기능을 알아야 한다.

👨‍💻 `크롤러` 이용 예

- `검색 엔진 인덱싱(search engine indexing)`: `크롤러`로 웹 페이지를 모아 검색 엔진을 위한 `로컬 인덱스(local index)`를 만든다.
  - `크롤러`의 가장 보편적인 용례
  - Googlebot: 구글(Google) 검색 엔진이 사용하는 `웹 크롤러`
- `웹 아카이빙(web archiving)`: 나중에 사용할 목적으로 장기보관하기 위해 웹에서 정보를 모은다.
  - 미국 국회 도서관(US Library of Congress), EU 웹 아카이브
- `웹 마이닝(web mining)`: 인터넷에서 유용한 지식을 도출해 낸다.
  - 유명 금융 기업들은 `크롤러`를 사용해 주주 총회 자료나 연차 보고서(annual report)를 다운받아 기업의 핵심 사업 방향을 알아낸다.
- `웹 모니터링(web monitoring)`: 인터넷에서 저작권이나 상표권이 침해되는 사례를 모니터링한다.
  - 디지마크(Digimarc) 사는 `웹 크롤러`를 사용해 해적판 저작물을 찾아내서 보고한다.

👩‍💻 좋은 `크롤러`가 갖추어야 하는 특성

- `규모 확장성(scalability)`: 오늘날 웹은 수십억 개의 페이지가 존재한다. 따라서 병행성(parallelism)을 활용하면 보다 효과적으로 웹 크롤링을 할 수 있다.
- `안정성(robustness)`: 비정상적 입력이나 환경(잘못 작성된 `HTML`, 아무 반응이 없는 서버, 장애, 악성 코드가 붙어 있는 링크 등)에 잘 대응할 수 있어야 한다.
- `예절(politeness)`: 수집 대상 웹 사이트에 짧은 시간 동안 너무 많은 요청을 보내면 안된다.
- `확장성(extensibility)`: 새로운 형태의 콘텐츠를 지원하기 쉬어야 한다. 새로운 형태의 콘텐츠(이미지 파일 등)를 크롤링하고자 할 때 전체 시스템을 새로 설계해야 하는 상황이 없어야 한다.

8.1. 1단계: 문제 이해 및 설계 범위 확정

> *웹 크롤러의 기본 알고리즘*

1. `URL` 집합이 입력으로 주어지면, 해당 `URL`들이 가리키는 모든 웹 페이지를 다운로드한다.
2. 다운받은 웹 페이지에서 `URL`들을 추출한다.
3. 추출된 `URL`들을 다운로드할 URL 목록에 추가하고 위의 과정을 처음부터 반복한다.

> *질문을 통해 요구사항을 알아내고 설계 범위를 좁힌다.*

👩‍💻 이 크롤러의 주된 용도는 무엇인가요? 검색 엔진 인덱스 생성용인가요? 아니면 데이터 마이닝? 아니면 그 외의 다른 용도가 있나요?

👨‍💻 **검색 엔진 인덱싱에 쓰일 것입니다.**

👩‍💻 매달 얼마나 많은 웹 페이지를 수집해야 하나요?

👩‍💻 **10억 개(1 billion)의 웹 페이지를 수집해야 합니다.**

👩‍💻 새로 만들어진 웹 페이지나 수정된 웹 페이지도 고려해야 하나요?

👩‍💻 그렇습니다.

👩‍💻 수집한 웹 페이지는 저장해야 합니까?

👩‍💻 네. 5년간 저장해 두어야 합니다.

👩‍💻 중복된 콘텐츠는 어떻게 해야 하나요?

👩‍💻 중복된 콘텐츠를 갖는 페이지는 무시해도 됩니다.

개략적 규모 추정

- 매달 10억 개의 웹 페이지를 다운로드한다.
- QPS=10억(1 billion)/30일/24시간/3600초=대략 400페이지/초
- 최대(Peak) QPS=2×QPS=800
- 웹 페이지의 크기 평균은 500k라고 가정하자.
  - 저장용량: 10억 페이지×500k=500TB/월
  - 5년간 보관할 경우 저장용량: 500TB×12개월×5년=30PB

> *계산이 끝나면 결과를 면접관과 점검하여 합의한 후 진행한다.*

8.2. 2단계: 개략적 설계안 제시 및 동의 구하기

8.2.1. 크롤러의 작업 흐름(worflow)

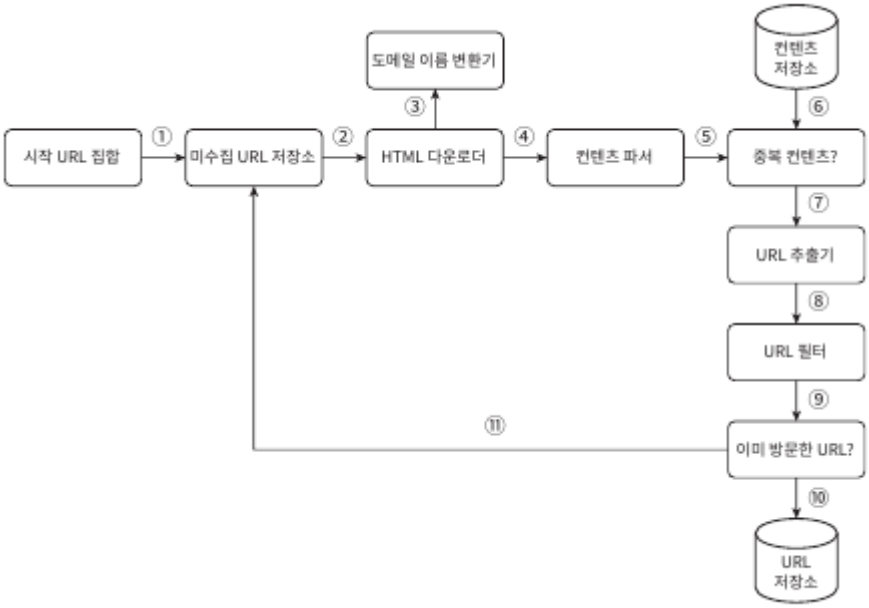


그림 9-4

웹 크롤러의 작업 흐름

1. `시작 URL`들을 `미수집 URL 저장소`에 저장한다.
2. `HTML 다운로더`는 `미수집 URL 저장소`에서 `URL` 목록을 가져온다.
3. `HTML 다운로더`는 `도메인 이름 변환기`를 사용하여 `URL`의 `IP 주소`를 알아내고, 해당 `IP 주소`로 접속하여 웹 페이지를 다운받는다.
4. `콘텐츠 파서`는 다운받은 `HTML` 페이지를 `파싱`하여 올바른 형식을 갖춘 페이지인지 `검증`한다.
5. `콘텐츠 파싱`과 `검증`이 끝나면 `중복 콘텐츠`인지 확인한다.
6. `중복 콘텐츠`인지 확인하기 위해 해당 페이지가 이미 `저장소`에 있는지 확인한다.
   - 이미 `저장소`에 있는 콘텐츠인 경우 처리하지 않고 버린다.
   - `저장소`에 없는 콘텐츠인 경우 `저장소`에 저장한 뒤 `URL 추출기`로 전달한다.
7. `URL 추출기`는 해당 `HTML` 페이지에서 링크를 골라낸다.
8. 골라낸 링크를 `URL 필터`로 전달한다.

9. 필터링이 끝나고 남은 URL만 중복 URL 판별 단계로 전달한다.
10. 이미 처리한 URL인지 확인하기 위하여, URL 저장소에 보관된 URL인지 확인하고 이미 저장소에 있는 URL은 버린다.
11. 저장소에 없는 URL은 URL 저장소에 저장하고 미수집 URL 저장소에 전달한다.

8.2.2. 각 컴포넌트의 기능

- 시작 URL 집합: 크롤링을 시작하는 출발점
  - 어떤 대학 웹사이트로부터 찾아 나갈 수 있는 모든 웹 페이지를 크롤링하는 직관적인 방법: 해당 대학의 도메인 이름이 붙은 모든 페이지의 URL을 시작 URL로 사용한다.
  - 전체 URL 공간을 작은 부분집합으로 나누어 주제별로 다른 시작 URL을 사용한다.
    - ex. URL 공간을 쇼핑, 스포츠, 건강 등의 주제별로 세분화하고 각각 다른 시작 URL을 사용한다.
- 미수집 URL 저장소(URL frontier): 다운로드 할 URL을 저장 관리하는 컴포넌트, FIFO(First-In-First-Out) 큐(Queue)
  - 대부분의 웹 크롤러는 크롤링 상태를 다운로드 할 URL, 다운로드한 URL 두 가지로 나누어 관리한다.
- HTML 다운로더(downloader): 인터넷에서 웹 페이지를 다운로드하는 컴포넌트
  - 다운로드 할 페이지의 URL은 미수집 URL 저장소가 제공한다.
- 도메인 이름 변환기: HTML 다운로더가 도메인 이름 변환기를 사용하여 URL에 대응되는 IP 주소를 알아낸다.
  - ex. www.wikipedia.org의 IP 주소: 198.35.26.96
- 콘텐츠 파서: 파싱(parsing)과 검증(validation) 절차
  - 이상한 웹 페이지는 문제를 일으킬 수 있고 저장 공간만 낭비한다.
  - 크롤링 서버 안에 콘텐츠 파서:를 구현하면 크롤링 과정이 느려질 수 있어 독립된 컴포넌트로 만든다.
- 중복 콘텐츠인가?: 29% 가량의 웹 페이지 콘텐츠는 중복이다. 이 문제를 해결하기 위한 자료 구조를 도입하여 데이터 중복을 줄이고 데이터 처리에 소요되는 시간을 줄인다.
  - 웹 페이지의 해시 값을 비교하여 두 HTML의 문서를 효율적으로 비교한다.
- 콘텐츠 저장소: HTML 문서를 보관하는 시스템

    - 저장할 데이터의 유형, 크기, 저장소 접근 빈도, 데이터의 유효 기간 등을 종합적으로 고려하여 저장소를 구현해야 한다.
    - 본 설계안의 경우, `디스크`와 `메모리`를 동시에 사용하는 저장소를 사용한다.
        - 데이터 양이 너무 많으므로 대부분의 콘텐츠는 `디스크`에 저장한다.
        - 인기 있는 콘텐츠는 `메모리`에 두어 접근 지연시간을 줄인다.
- `URL 추출기`: `HTML` 페이지를 파싱하여 링크들을 골라낸다.

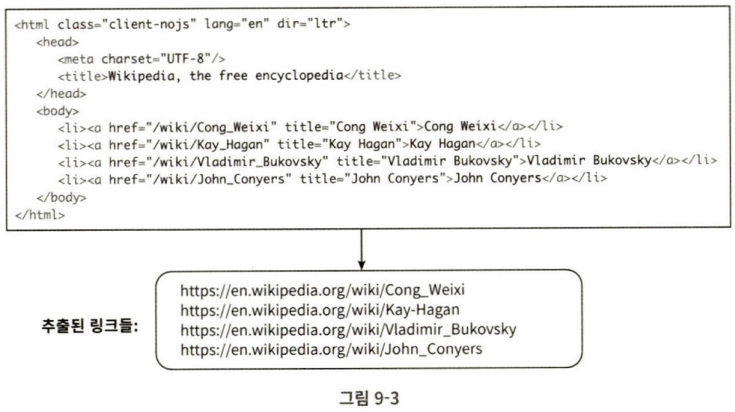


그림 9-3

상대경로를 절대경로로 변환하여 링크를 추출한다.

- `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 배제한다.
- `이미 방문한 URL인가?`: 자료 구조를 사용하여 이미 방문한 적이 있는 `URL`인지 추적하여 같은 `URL`을 여러 번 처리하는 일을 방지한다. → 서버 부하를 줄이고 시스템이 무한 루프에 빠지는 일을 방지할 수 있다.
    - 자료 구조: `블룸 필터(bloom filter)`, `해시 테이블`
- `URL 저장소`: 이미 방문한 URL을 저장하는 저장소

8.3. 3단계: 상세 설계

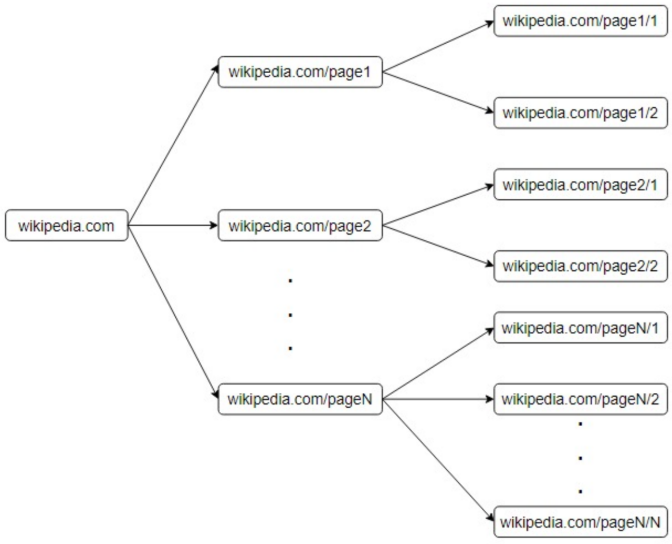
8.3.1. DFS(Depth-First Search) vs BFS(Breath-First Search)

- `크롤링 프로세스`는 `유향 그래프(directed graph)`를 `에지(edge)`를 따라 탐색하는 과정이다.
    - 웹: `유향 그래프(directed graph)`

- 페이지: `노드`
- 하이퍼링크(URL): `에지(edge)`

- 그래프 탐색에 널리 사용되는 알고리즘: `깊이 우선 탐색법(Depth-First Search, DFS)`, `너비 우선 탐색법(Breath-First Search, BFS)`
  - `DFS`는 그래프 크기가 클 경우 어느 정도로 깊숙이 가게 될지 가늠하기 어려워 좋은 선택이 아닐 수 있다.

- `웹 크롤러`는 보통 `BFS`를 사용한다.
  - `BFS`는 `FIFO(First-In-First-Out) 큐`를 사용하는 알고리즘이다.
  - 한 쪽으로 탐색할 `URL`을 집어넣고, 다른 한쪽으로 꺼낸다.
  - `FIFO(First-In-First-Out) 큐` 구현의 문제점
    - 한 페이지에서 나오는 링크의 대부분이 같은 서버로 되돌아간다.
      - ex. `크롤러`는 같은 호스트에 속한 많은 링크를 다운받느라 바빠지는데, 링크를 병렬로 처리하면 위키피디아 서버는 수많은 요청으로 과부하가 걸린다. → `예의 없는(impolite)` `크롤러`



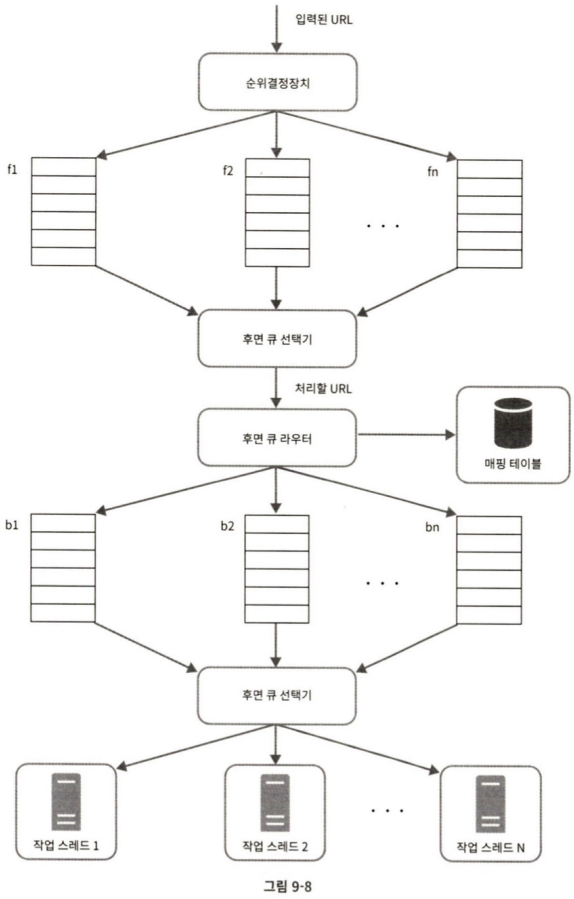
wikipedia.com 페이지에서 추출한 모든 링크는 내부 링크, 즉 동일한 wikipedia.com 서버의 다른 페이지를 참조하는 링크이다.

    - 표준적인 `BFS` 알고리즘은 `URL` 간에 우선순위를 두지 않는다.(처리 순서에 있어 모든 페이지를 공평하게 대우한다.)
      - 모든 웹 페이지가 같은 수준의 품질, 같은 수준의 중요성을 갖지 않는다.

- 페이지 순위(page rank), 사용자 트래픽의 양, 업데이트 빈도 등 여러가지 척도에 비추어 처리 우선순위를 구별해야 한다.

8.3.2. 미수집 URL 저장소

- `미수집 URL 저장소`를 활용하면 `예의(politeness)`를 갖춘 `크롤러`, `URL` 사이의 우선순위와 `신선도(freshness)`를 구별하는 `크롤러`를 구현할 수 있다.



전면 큐(front queue): 우선순위 결정 과정을 처리
후면 큐(back queue): 크롤러가 예의 바르게 동작하도록 보증

- `예의`: 웹 크롤러는 수집 대상 서버로 짧은 시간 안에 너무 많은 요청을 보내는 것(`impolite`)을 삼가야 한다. → `Dos(Denial-of-Service) 공격`으로 간주되기도 한다.
  - 웹사이트의 호스트명(hostname)과 다운로드를 수행하는 작업 스레드(worker thread) 사이의 관계를 유지하여 **동일 웹 사이트에 대해서는 한 번에 한 페이지만 요청한다.**
    - 각 다운로드 스레드는 별도 `FIFO 큐`를 가지고 있어 해당 `큐`에서 꺼낸 `URL`만 다운로드한다.

    - `큐 라우터(queue router)`: 같은 호스트에 속한 `URL`은 언제나 같은 `큐(b1, b2, …,bn)`로 가도록 보장하는 역할을 한다.
    - `매핑 테이블(mapping table)`: 호스트 이름과 `큐` 사이의 관계를 보관하는 테이블

Host	Queue
wikipedia.com	b1
apple.com	b2
…	…
nike.com	bn

    - `FIFO 큐(b1부터 bn까지)`: 같은 호스트에 속한 `URL`은 언제나 같은 `큐`에 보관된다.
    - `큐 선택기(queue selector)`: `큐`들을 순회하면서 `큐`에서 `URL`을 꺼내 해당 `큐`에서 나온 `URL`을 다운로드하도록 지정된 직접 스레드에 전달한다.
    - `작업 스레드(worker thread)`: 전달된 `URL`을 다운로드하는 작업을 순차적으로 수행한다. 작업들 사이에 일정한 `지연시간(delay)`을 둘 수 있다.
- `우선순위`: `크롤러` 입장에서 중요한 페이지를 먼저 수집한다.
    - 우선순위 척도: 페이지랭크(PageRank), 트래픽 양, 갱신 빈도(update frquency) 등
    - `순위 결정 장치(prioritizer)`: `URL`을 입력 받아 `우선순위`를 계산한다.
    - `큐(f1, …, fn)`: 우선순위별로 큐가 하나씩 할당된다. 우선순위가 높으면 선택될 확률이 높아진다.
    - `큐 선택기`: 임의 큐에서 처리한 `URL`을 꺼내는 역할(순위가 높은 큐에서 더 자주 꺼내도록 프로그램)
- `신선도(freshness)`: 이미 다운로드한 페이지를 주기적으로 재수집(recrawl)해야 한다.
    - 최적화 방법
        - 웹 페이지의 변경 이력(update history)를 활용한다.
        - 우선순위를 활용하여 중요한 페이지는 좀 더 자주 재수집한다.
- `미수집 URL 저장소`를 위한 `지속성 저장장치`

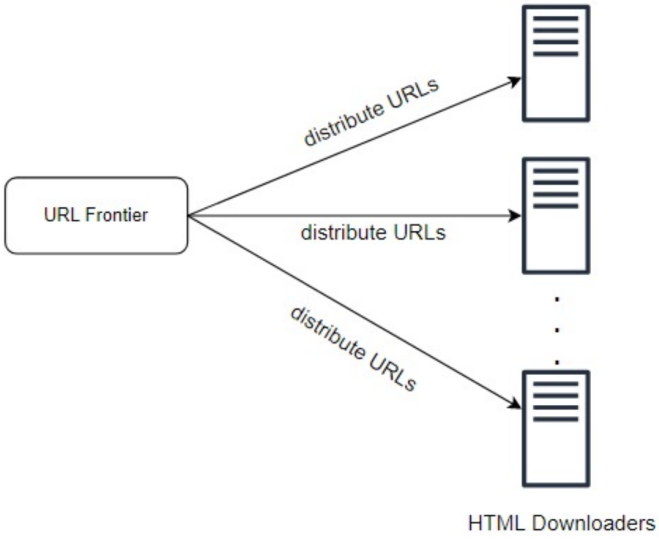
- 검색 엔진을 위한 `크롤러`의 경우 처리해야 하는 `URL`의 수가 너무 많아 모두 메모리에 보관하는 것은 `안정성`이나 `규모 확장성` 측면에서 바람직하지 않다.
  - 전부 디스크에 저장하면 쉽게 `성능 병목지점`이 된다.
  - 대부분의 `URL`은 `디스크`에 두고 IO 비용을 줄이기 위해 `메모리 버퍼`에 `큐`를 둔다.
    - `버퍼`에 있는 데이터는 주기적으로 `디스크`에 기록한다.

8.3.3. HTML 다운로더

- `HTML 다운로더(downloader)`는 `HTTP 프로토콜`을 통해 웹 페이지를 다운받는다.
- `Robots.txt`
  - 크롤러가 수집해도 되는 페이지 목록이 들어있다.

```
User-agent: Googlebot
Disallow:/creatorhub/*
Disallow:/rss/people/*/reviews
Disallow:/gp/pdp/rss/*/reviews
Disallow:/gp/cdp/member-reviews
Disallow:/gp/aw/c/r
```

  - 이 파일을 계속 다운로드하지 않기 위해 주기적으로 다운받아 `캐시`에 보관한다.
- **성능 최적화 기법**
  - `분산 크롤링`: 성능을 높이기 위해 `크롤링` 작업을 여러 서버에 분산한다.
    - 각 서버는 여러 스레드를 돌려 다운로드 작업을 처리한다.



URL 공간을 작은 단위로 분할하여 각 서버는 일부의 다운로드를 담당한다.

- `도메인 이름 변환 결과 캐시`: DNS 조회 결과로 얻어진 `도메인` 이름과 `IP 주소` 사이의 관계를 `캐시`에 보관해 놓고 `크론 잡(cron job)` 등을 돌려 주기적으로 갱신한다.
  - `도메인 이름 변환기(DNS Resolver)`는 `크롤러` 성능의 병목 중 하나이다.
    - DNS 요청을 보내고 결과를 받는 작업을 한다.(`동기`) DNS 요청의 결과(10ms-200ms)를 받기 전에는 다음 작업을 진행할 수 없다.
    - `크롤러` 스레드 중 DNS 요청을 보내고 결과를 받는 작업을 하고 있으면 다른 스레드의 DNS 요청은 전부 `블록(block)`된다.
- `지역성`: `크롤링` 작업을 수행하는 서버를 지역별로 분산한다.
  - `크롤링 서버`가 `크롤링` 대상 서버와 지역적으로 가까우면 페이지 다운로드 시간이 줄어든다.
  - `크롤 서버`, `캐시`, `큐`, `저장소` 등 대부분의 컴포넌트에 적용할 수 있다.
- **짧은** `타임아웃`: 최대 얼마나 기다릴지 미리 정해둔다. `타임아웃` 시간 동안 서버가 응답하지 않으면 크롤러는 해당 페이지 다운로드를 중단하고 다음 페이지로 넘어간다.
  - 어떤 웹 서버는 응답이 느리거나 아예 응답하지 않는다. 이럴 경우 `대기 시간(wait time)`이 길어진다.

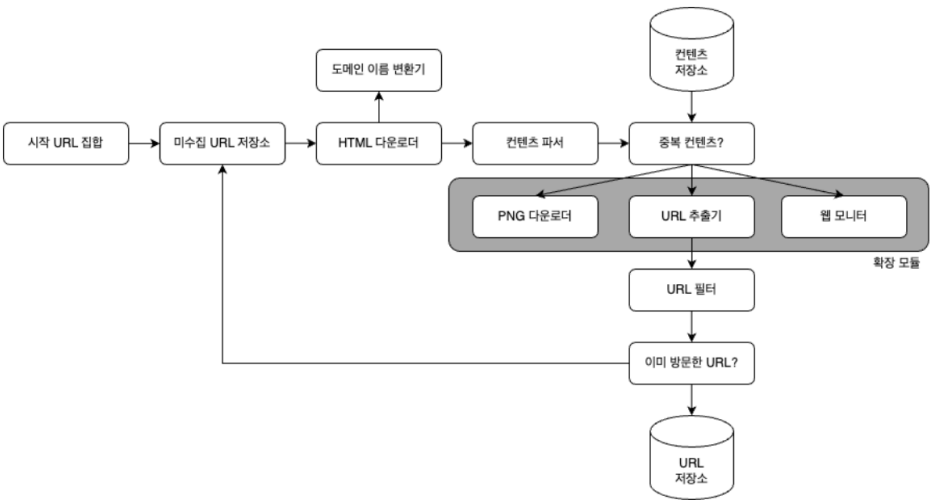
8.3.4. 안정성 확보 전략

> *`안정성`을 향상시키는 방법*

- `안정 해시(consistent hashing)`: 다운로더 서버들에 부하를 분산할 수 있고 다운로더 서버를 쉽게 추가하고 삭제할 수 있다.
- `크롤링` 상태 및 수집 데이터 저장: 장애가 발생한 경우에도 쉽게 복구할 수 있도록 `크롤링` 상태와 수집된 데이터를 지속적 저장장치에 기록한다.
    - 저장된 데이터를 로딩하면 중단되었던 크롤링을 쉽게 재시작할 수 있다.
- `예외 처리(exception handling)`: 예외가 발생해도 전체 시스템이 중단되는 일 없이 그 작업을 할 수 있어야 한다.
- `데이터 검증(data validation)`: 시스템 오류를 방지하기 위한 중요 수단이다.

8.3.5. 확장성 확보 전략

- 새로운 형태의 콘텐츠를 쉽게 지원할 수 있어야 한다.
    - 새로운 모듈을 끼워 넣어 새로운 형태의 콘텐츠를 지원할 수 있도록 설계해 보자.



    - PNG 다운로더: PNG 파일을 다운로드하는 플러그인(plugin) 모듈
    - 웹 모니터(web monitor): 웹을 모니터링하여 저작권이나 상표권이 침해되는 일을 막는 모듈

8.3.6. 문제 있는 콘텐츠 감지 및 회피 전략

> *중복이거나 의미 없는, 유해한 콘텐츠를 감지하고 시스템으로부터 차단하는 방법*

- `중복 콘텐츠`: `해시`나 `체크섬(check-sum)`을 사용하여 중복 콘텐츠를 쉽게 탐지할 수 있다.
- `거미 덫(spider trap)`: `크롤러`를 무한 루프에 빠뜨리도록 설계한 웹 페이지
  - ex. spidertrapexample.com/foo/bar/foo/bar/foo/bar/…
  - `URL`의 최대 길이를 제한하여 회피할 수 있다.
  - 사람이 수작업으로 덫을 확인하고 찾아낸 후 덫이 있는 사이트를 `크롤러` 탐색 대상에서 제외하거나 `URL` 필터 목록에 저장해둔다.
- `데이터 노이즈`: 가치가 없는 콘텐츠를 제외한다.
  - ex. 광고, 스크립트 코드, 스팸 `URL`

8.4. 4단계: 마무리

👨‍💻 추가 논의 사항

- `서버 측 렌더링(server-side rendering)`: 페이지를 `파싱`하기 전에 `서버 측 렌더링(동적 렌더링, dynamic rendering)`을 적용한다.
  - 많은 웹사이트가 `자바스크립트(Java Script)`, `AJAX` 등의 기술을 사용해서 링크를 즉석에서 만들어 낸다. → 웹페이지를 그대로 다운받아서 파싱하면 동적으로 생성되는 링크를 발견할 수 없다.
- 원치 않는 페이지 `필터링`: 스팸 방지(anti-spam) 컴포넌트를 두어 품질이 조악하거나 스팸성인 페이지를 걸러낸다.
  - 저장 공간 등 `크롤링`에 소요되는 자원은 유한하다.
- `데이터베이스 다중화(replication) 및 샤딩(sharding)`: `데이터 계층(data layer)`의 `가용성`, `규모 확장성`, `안정성`이 확장된다.
- `수평적 규모 확장성(horizontal scalability)`: 대규모 `크롤링`을 위해서 다운로드를 실행할 서버가 여러 대 필요할 수 있다.
  - 서버가 상태 정보를 유지하지 않도록 하는 것이 중요하다.(`무상태(stateless) 서버`로 만든다.)
- `가용성`, `일관성`, `안정성`
- `데이터 분석 솔루션(analytics)`