5장. 안정 해시 설계

제이(jay-so)

목차

1. 노드 기반의 캐시 서버

해시 함수란?

- 사전적 정의로 "임의의 길이를 가진 입력 데이터(메시지)를 고정된 길이의 출력(해시 값 또는 해시 코드)으로 매핑하는 함수를 의미"

- 노션 - 해시 정리 페이지

1. 노드 기반의 캐시 서버

- N개의 캐시 서버가 존재
- 부하를 균등하게 나누기 위해 다음의 해시 함수를 사용한다고 가정

서버 인덱스(Server Index) = Hash(key) % N(서버의 갯수)

키	해시	해시 % 4 (서버 인덱스)
key0	18358617	1
key1	26143584	0
key2	18131146	2
key3	35863496	0
key4	34085809	1
key5	27581703	3
key6	38164978	2
key7	22530351	3

표 5-1

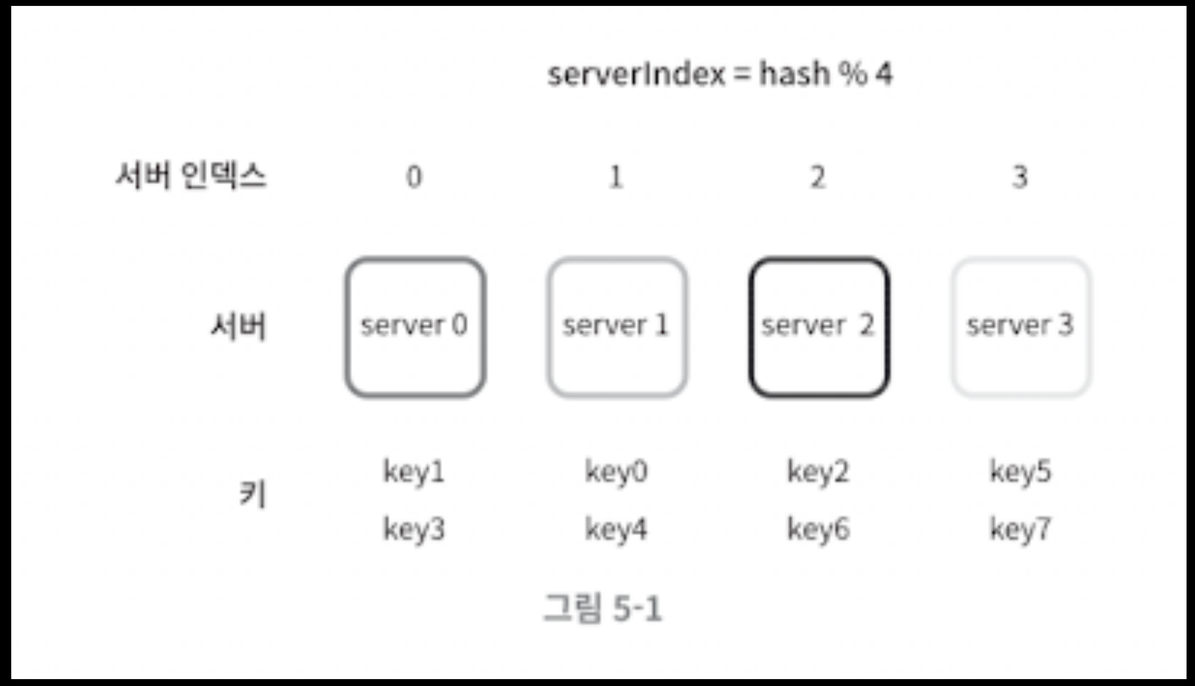


그림 5-1

1. 노드 기반의 캐시 서버

문제점

- 서버가 추가되거나 기존 서버가 삭제될 경우 문제 발생
- 예시) - 1번 서버가 장애가 난 경우(서버 풀 = 3으로 변동)

키	해시	해시 % 3 (서버 인덱스)
key0	18358617	0
key1	26143584	0
key2	18131146	1
key3	35863496	2
key4	34085809	1
key5	27581703	0
key6	38164978	1
key7	22530351	0

표 5-2

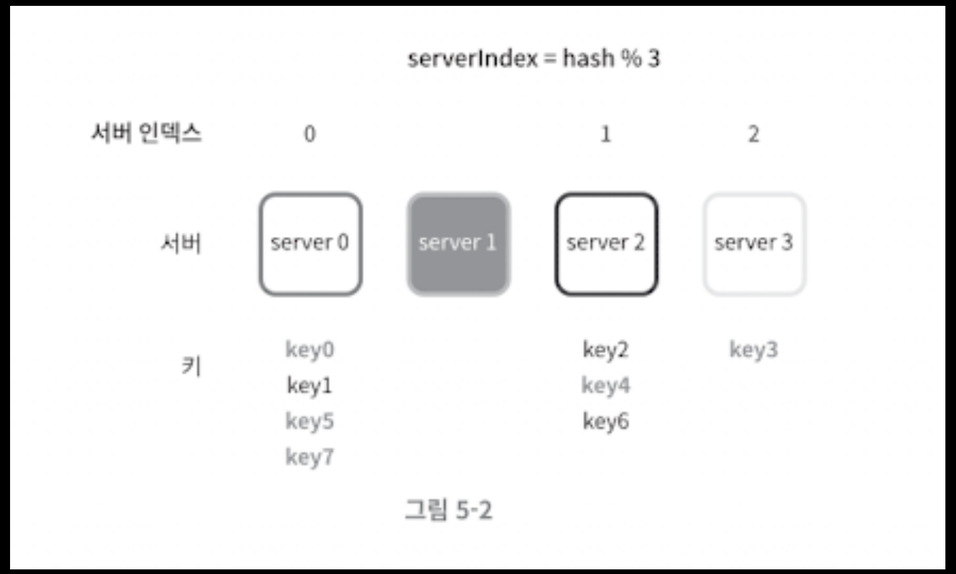
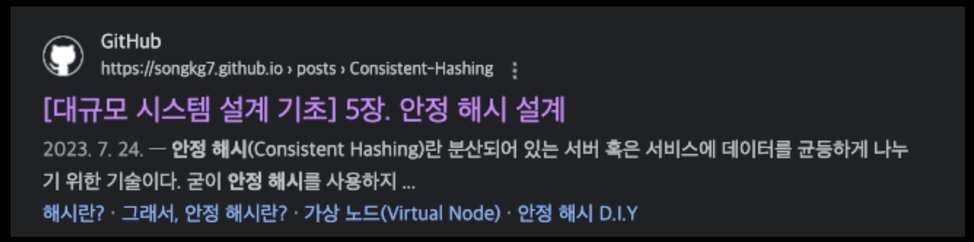


그림 5-2

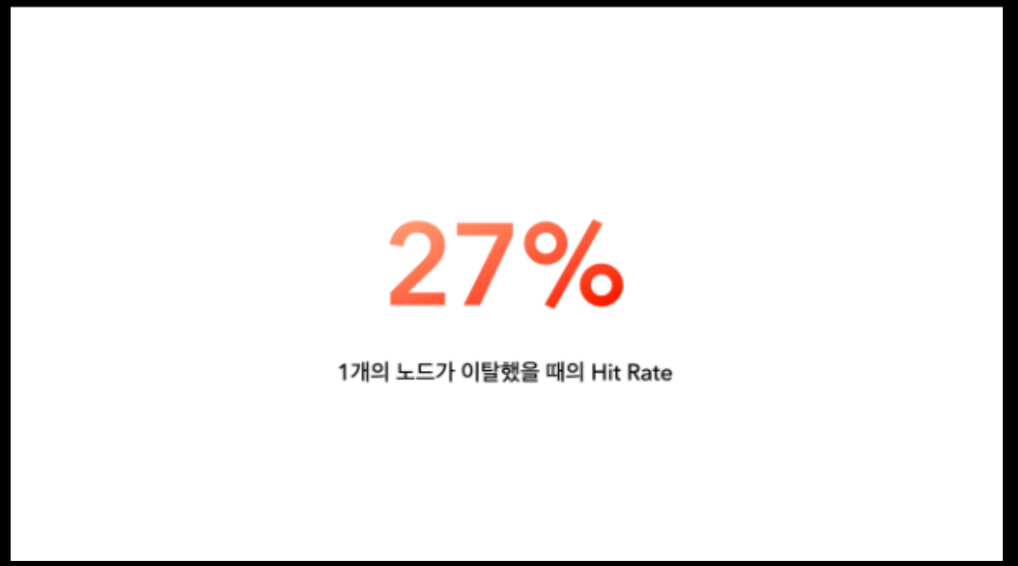
1. 노드 기반의 캐시 서버

- 검증해본 예시 (블로그 참고)

- 4개의 노드로 실험해본 결과. 1개 노드만 이탈해도 캐시 적중률이 27% 로 급격히 하락



https://songkg7.github.io/posts/Consistent-Hashing/



2. 안정 해시

- 정의

- 해시 테이블 크기가 조정될 때, 평균적으로 오직 K/N개의 키만 재배치하는 해시 기술

- 해시 공간과 해시 링

- 해시 공간의 양쪽을 구부려 접으면, 다음과 같은 해시 링이 생성됨

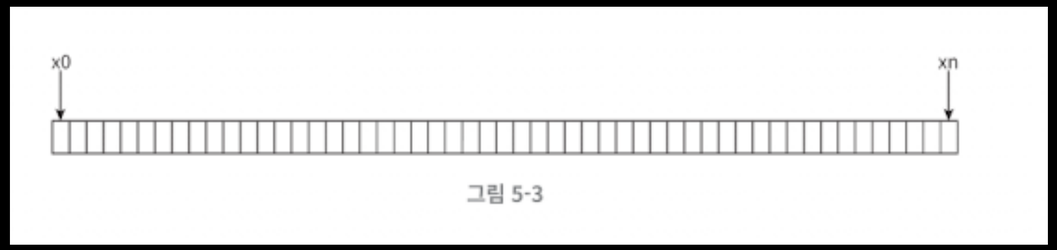


그림 5-3

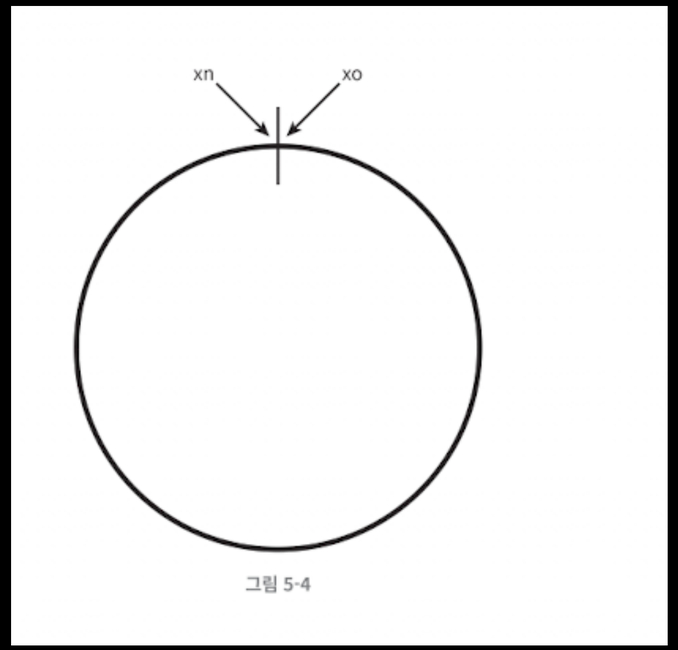


그림 5-4

2. 안정 해시

- 정의

- 해시 공간과 해시 링

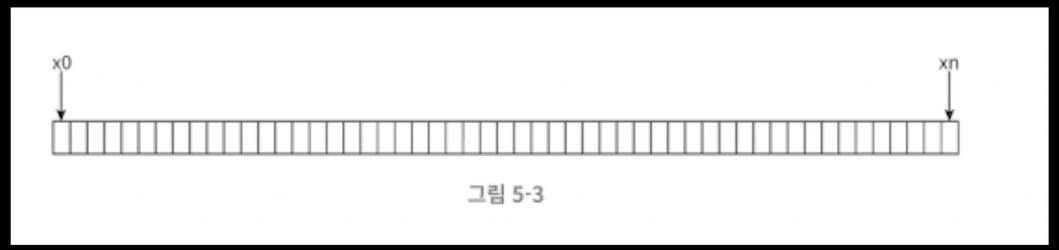


그림 5-3

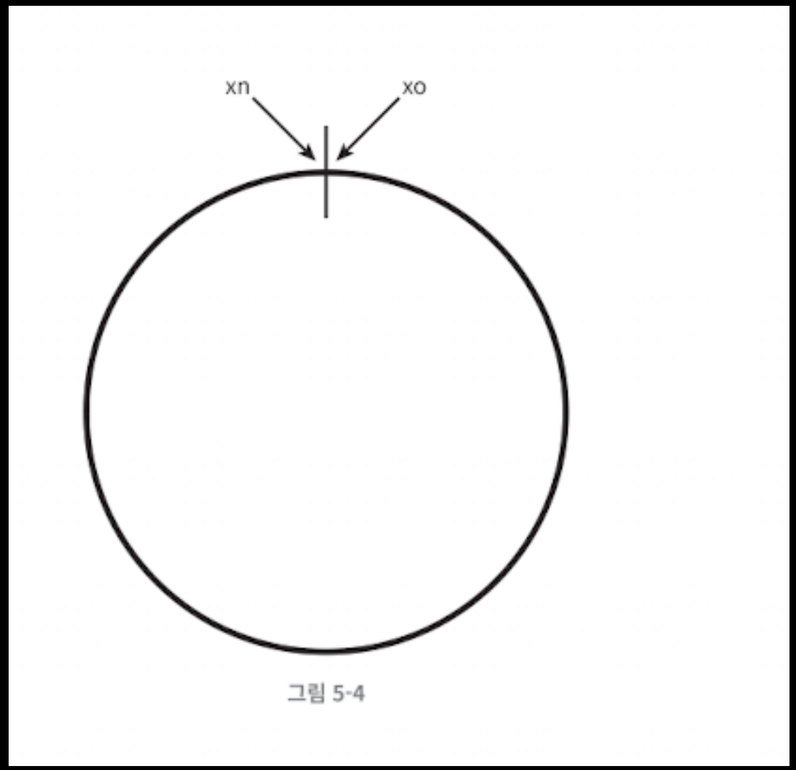


그림 5-4

2. 안정 해시

-해시 서버

- 해시 함수 f를 사용하면서 서버 IP나 이름을 배치한 결과

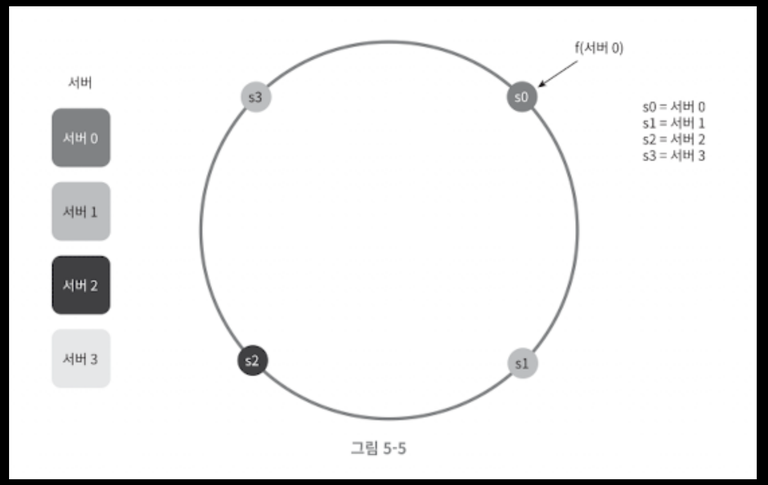


그림 5-5

- 해시 키

- 캐시할 키 key0,key1,key3 또한 링 위의 어느 지점에 배치할 수 있음

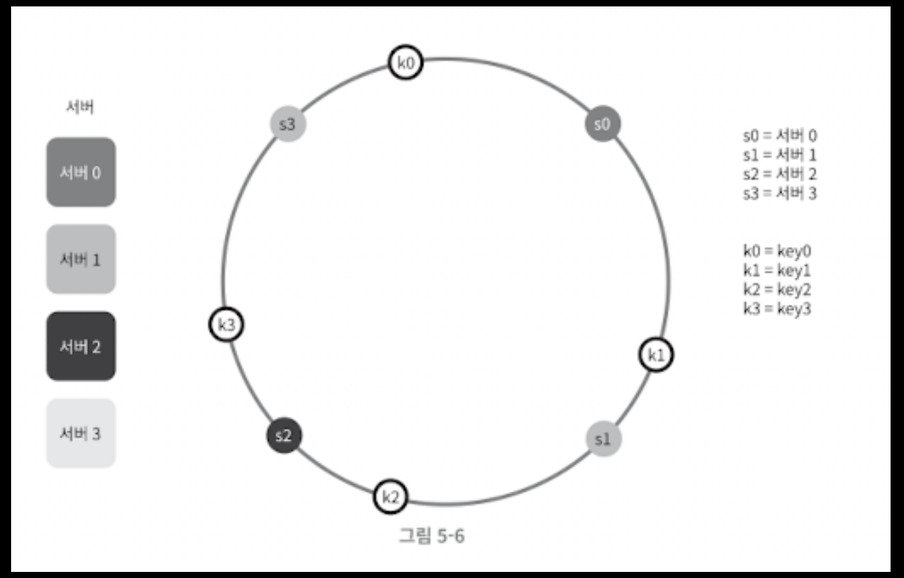


그림 5-6

2. 안정 해시

-해시 서버

- 해시 함수 f를 사용하면서 서버 IP나 이름을 배치한 결과

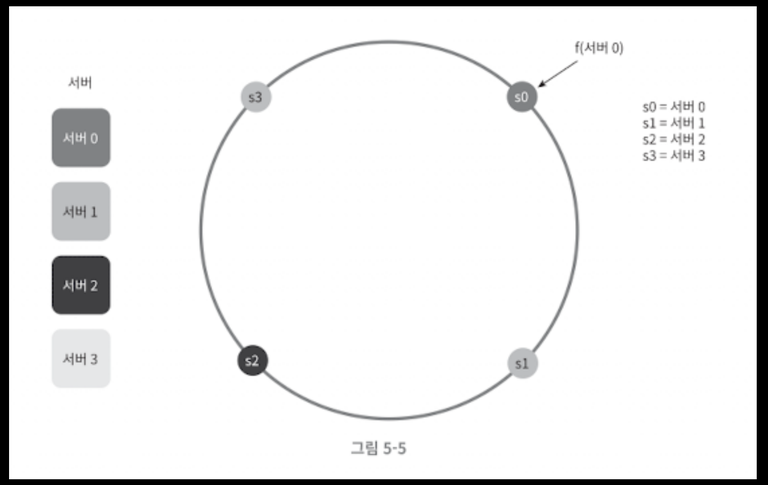


그림 5-5

- 해시 키

- 캐시할 키 key0,key1,key3 또한 링 위의 어느 지점에 배치할 수 있음

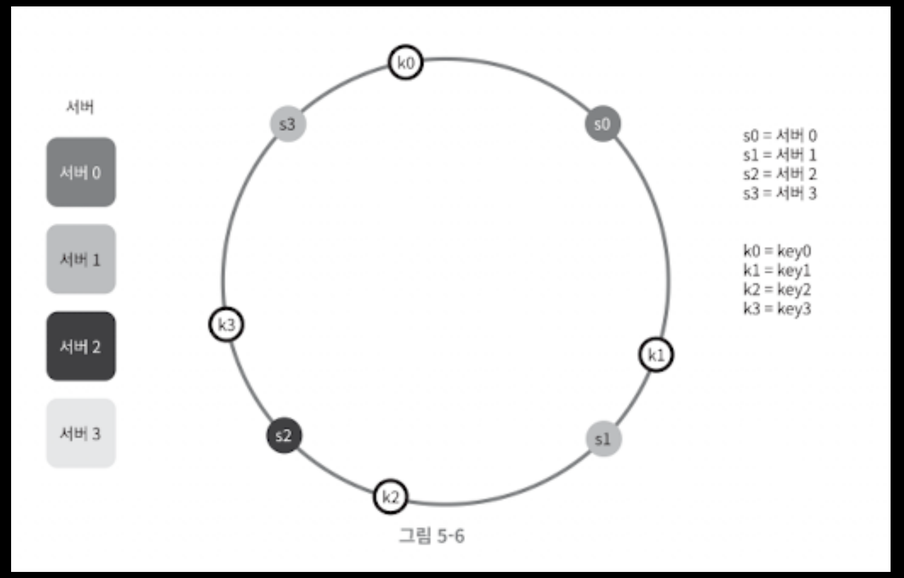


그림 5-6

2. 안정 해시

- 서버 조회

- 캐시 키가 저장되는 서버는 해당 키의 위치로부터, 시계방향으로 링을 돌아가면서 만나게 되는 첫번째 서버

- Key0 = 서버 0에 저장됨
- Key1 = 서버 1에 저장됨
- Key2 = 서버 2에 저장됨
- Key3 = 서버 3에 저장됨

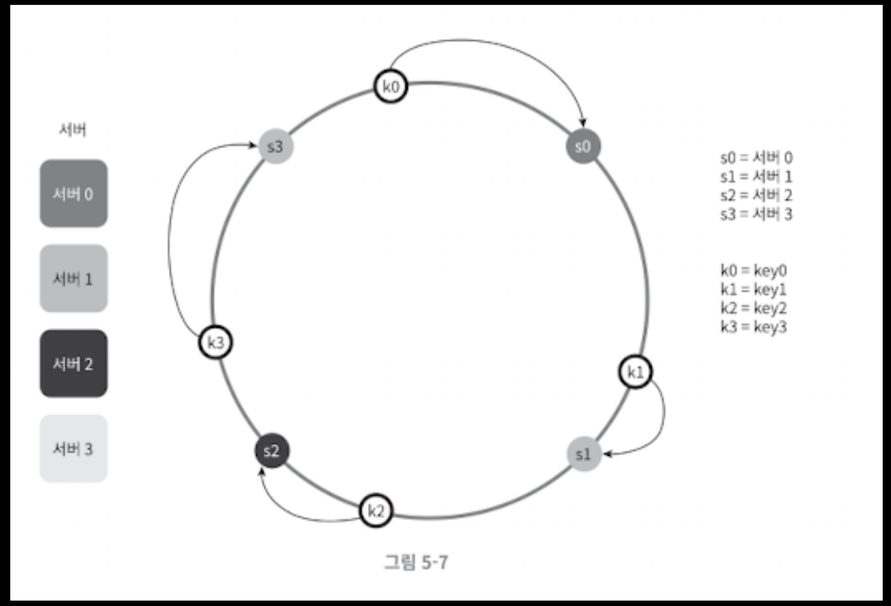


그림 5-7

- 서버 추가

- 서버 4가 추가되면, 해당 서버 직전의 키 값만 재배치하면 됨
- Key 0 -> 서버 0에 저장 -> 서버 4추가 -> key0가 서버 4에 저장됨 -> 재배치

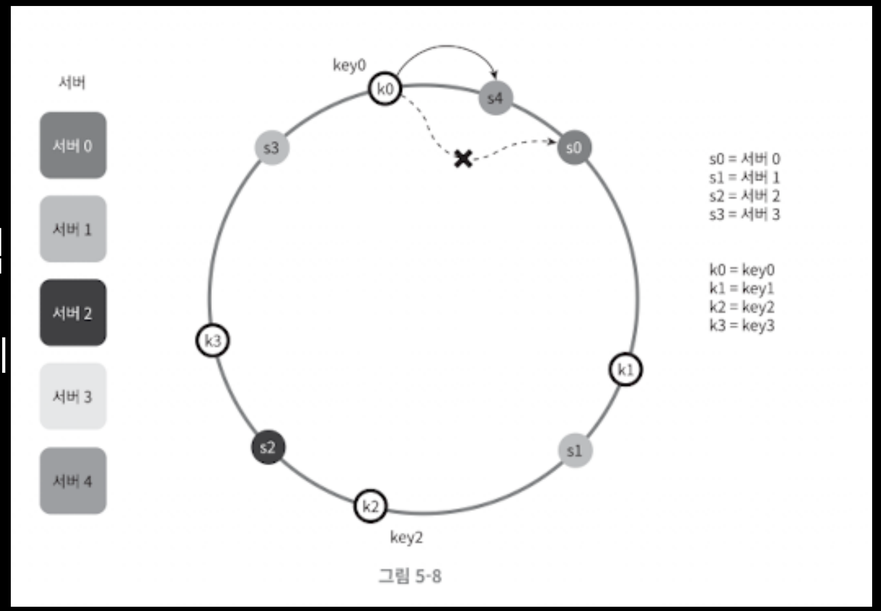


그림 5-8

2. 안정 해시

- 서버 조회

- 서버 추가

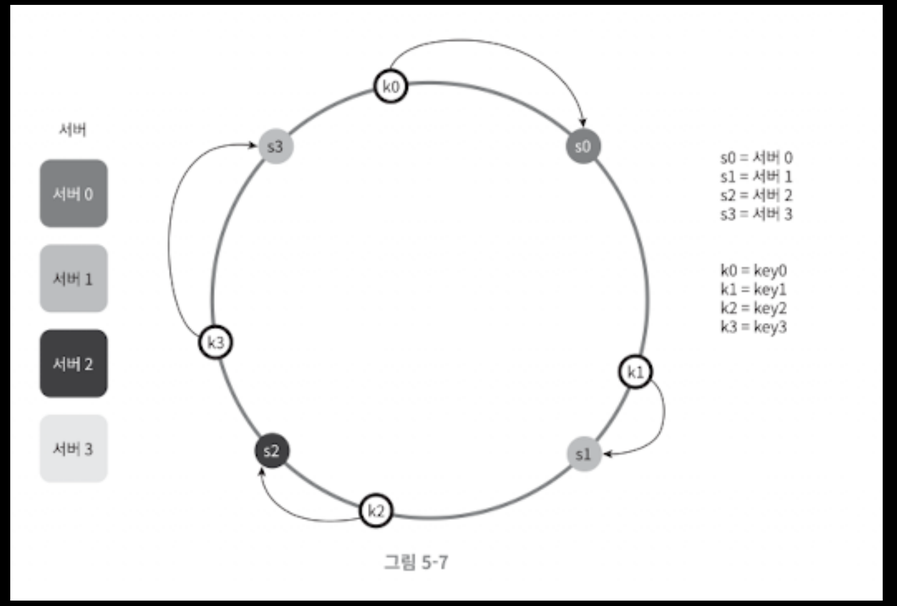


그림 5-7

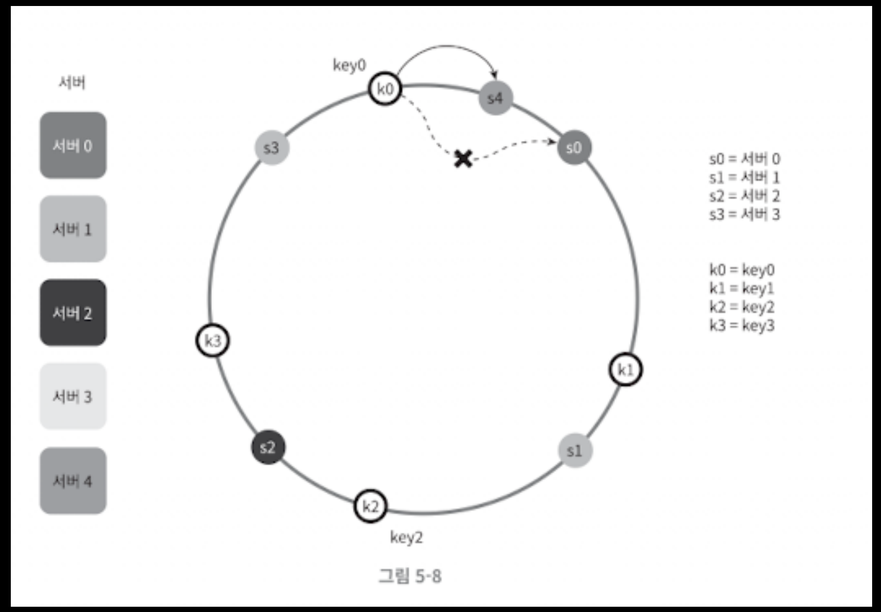


그림 5-8

2. 안정 해시

- 서버 제거

- 서버 하나가 제거되면, 해당 서버 이전의 키 만 재배치되면 됨
  - 서버1이 삭제될 때, key1만 서버 2로 재배치됨

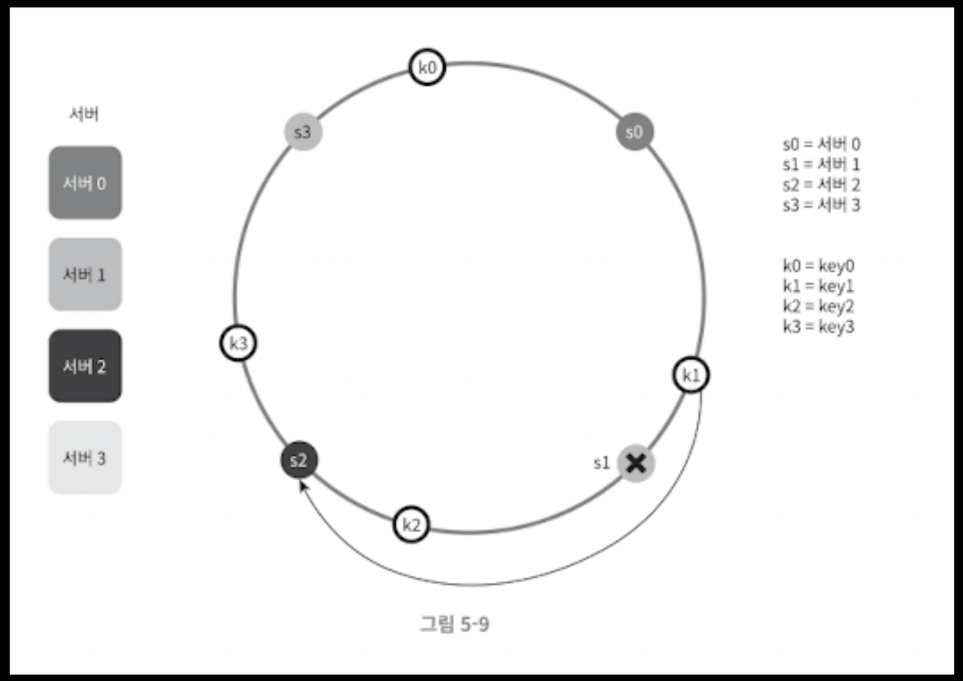


그림 5-9

2. 안정 해시

- 2가지 문제점

- 안정 해시 알고리즘의 기본 컨셉(2가지)

1. 서버와 키를 균등 분포 해시 함수를 사용하여 해시 링에 배치함

2. 키의 위치에서 링을 시계 방향으로 탐색하다 만나는 최초의 서버가 키가 저장될 서버가 됨

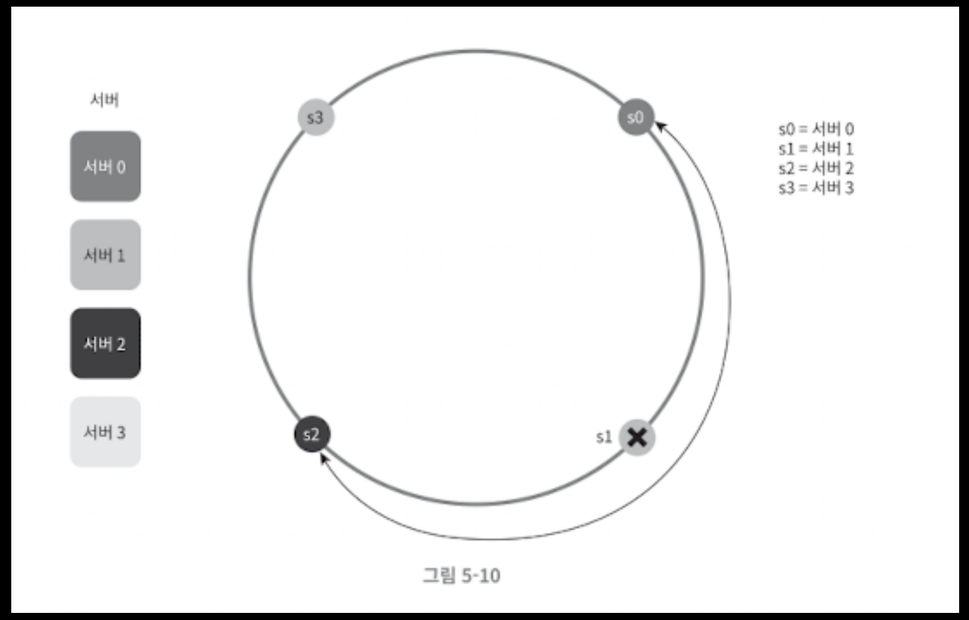


그림 5-10

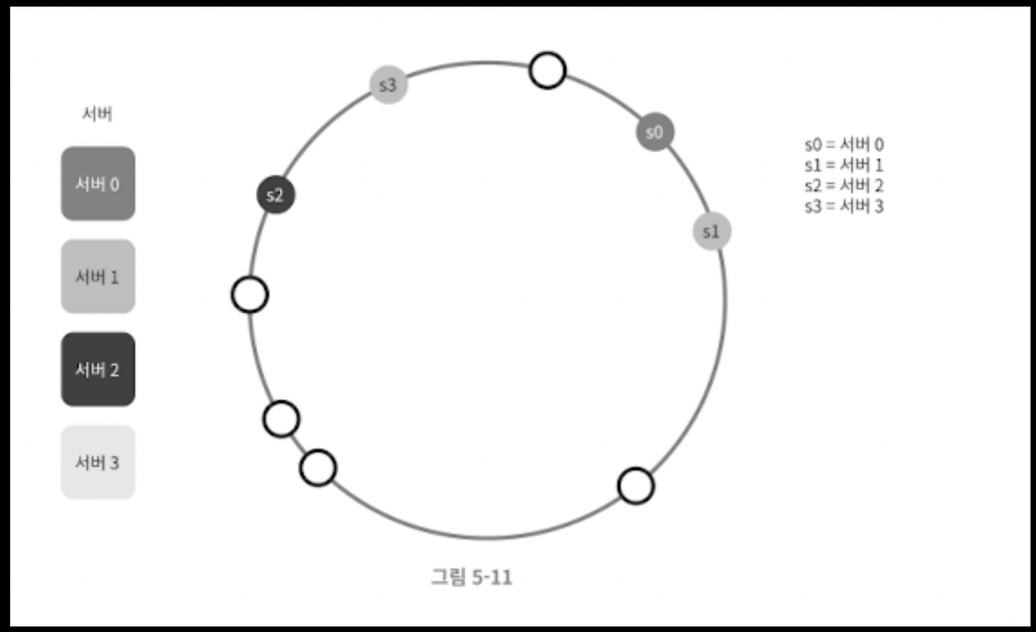


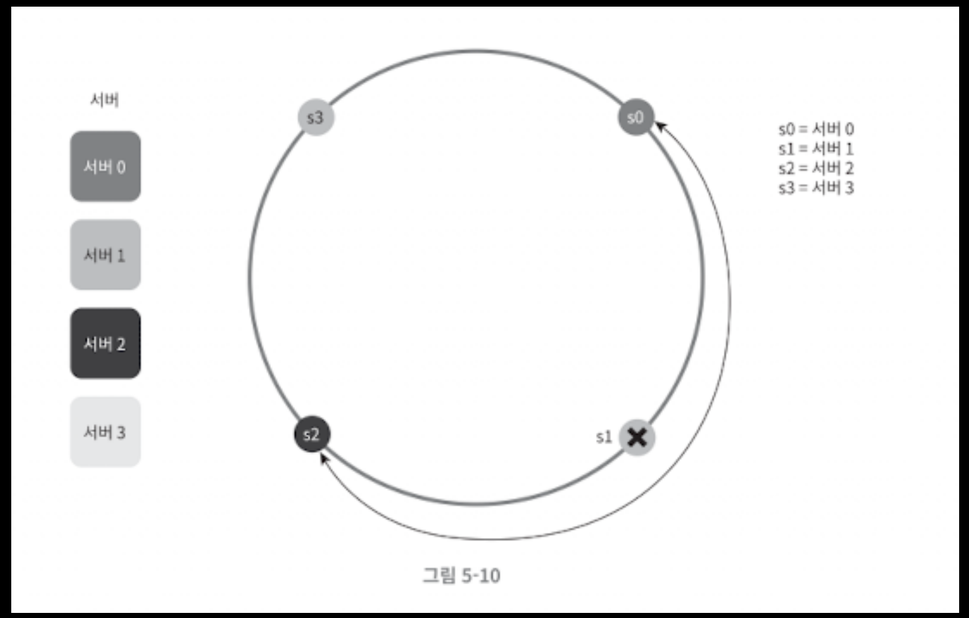
그림 5-11

2. 안정 해시

- 2가지 문제점

1.서버가 추가되거나, 삭제되는 상황에서 파티션의 크기를 균등하게 유지하는 것이 불가능

파티션 = 인접한 서버 사이의 해시 공간
s1이 삭제되면, s2의 파티션이 다른 파티션 대비 거의 2배로 커짐



2. 안정 해시

- 2가지 문제점

  2. 키의 균등 분포를 달성하기 어려움

     서버 1과 서버 3은 아무 데이터도 갖지 않는 반면, 대부분의 키는 서버2에 보관됨

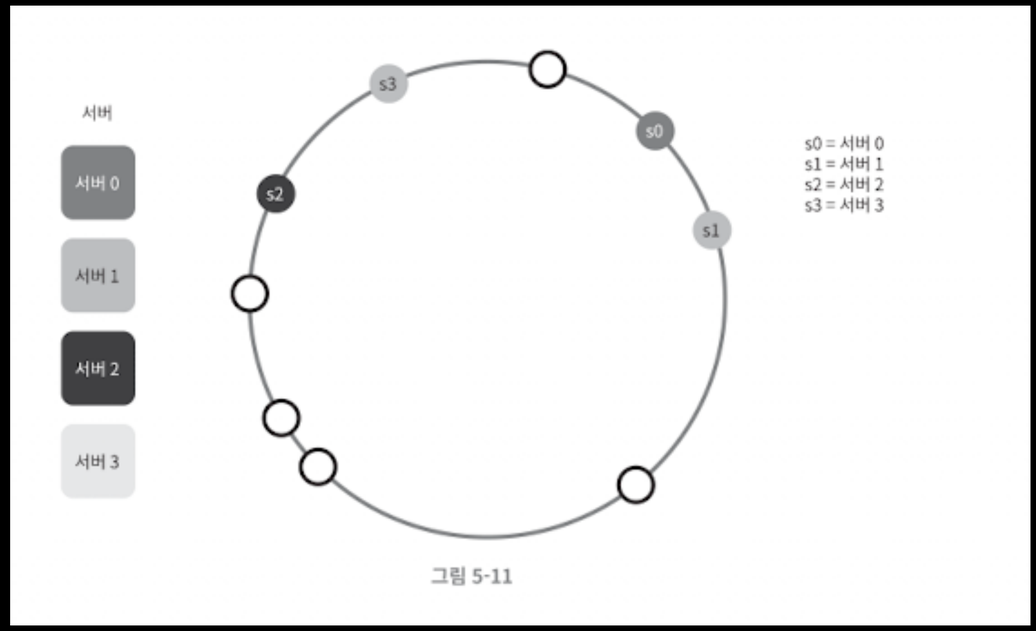


그림 5-11

2. 안정 해시

- 가상 노드

안정 해시의 구현 문제점 2가지에 대해서 해결책으로 등장

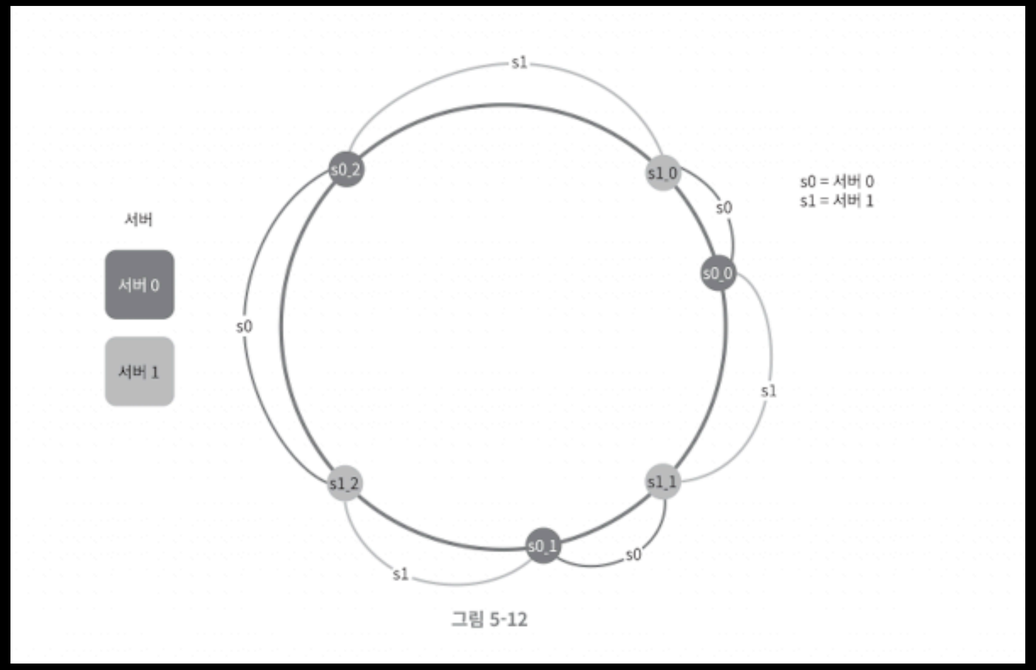


그림 5-12

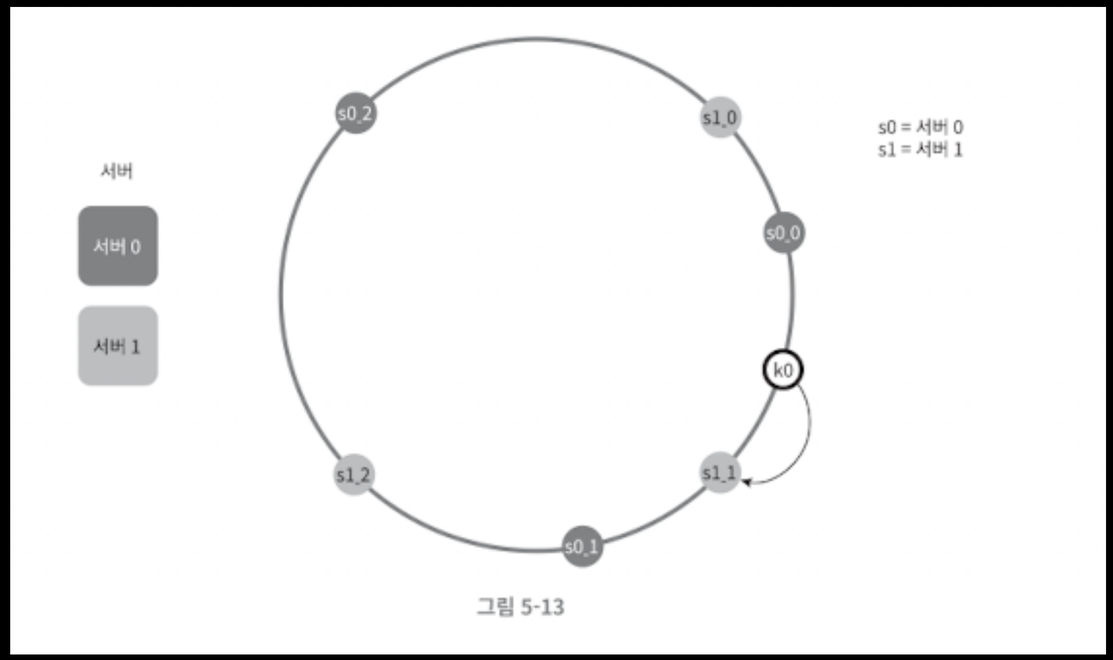


그림 5-13

2. 안정 해시

- 가상 노드

- 실제 노드 또는 서버를 가리키는 노드
- 하나의 서버는 링 위에 여러 개의 가상 노드를 가질 수 있음

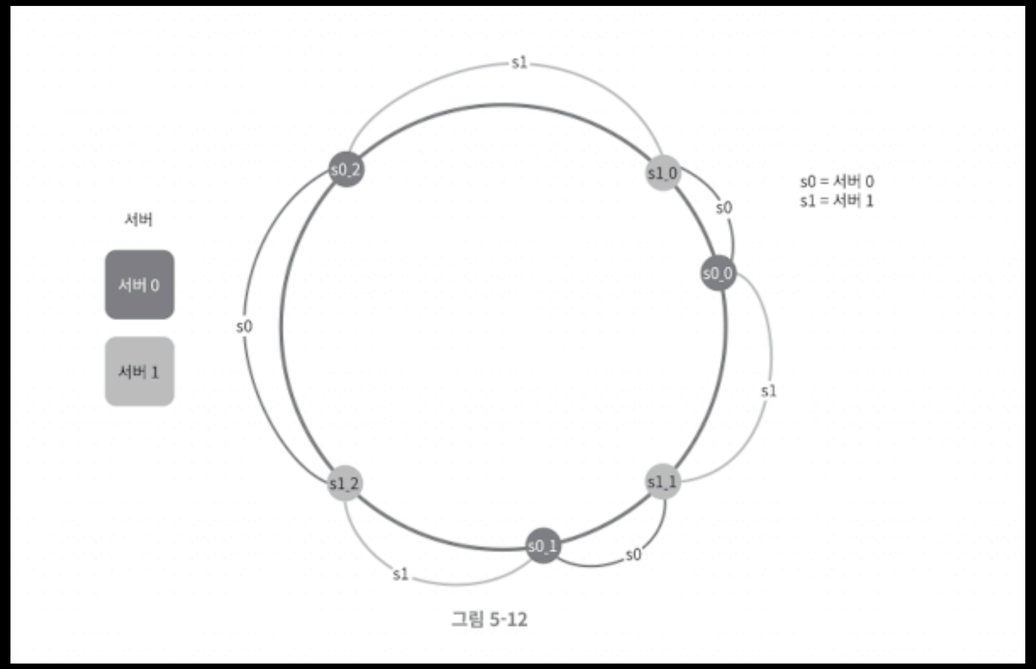


그림 5-12

- 서버 0과 서버 1은 3개의 가상 노드를 가짐
- 각 서버는 하나가 아닌 여러 개의 파티션을 관리해야 함

2. 안정 해시

- 가상 노드

키의 위치로부터 시계 방향으로 링을 탐색하다 만나는 최초의 가상 노드가 해당 키가 저장될 서버가 됨

- k0가 저장되는 서버 = k0의 위치로부터 링을 시계 방향으로 탐색하다 만나는 최초의 가상 노드 s1_1가 되는 서버, 즉 서버 1임

- 가상 노드의 갯수를 늘리면 키의 분포는 점점 균등해짐

- 표준 편차가 작아져 데이터가 고르게 분포됨

- 다만, 가상 노드를 늘리면, 가상 노드 공간 데이터를 저장할 공간은 더 필요함 -> 적절한 타협이 필요함

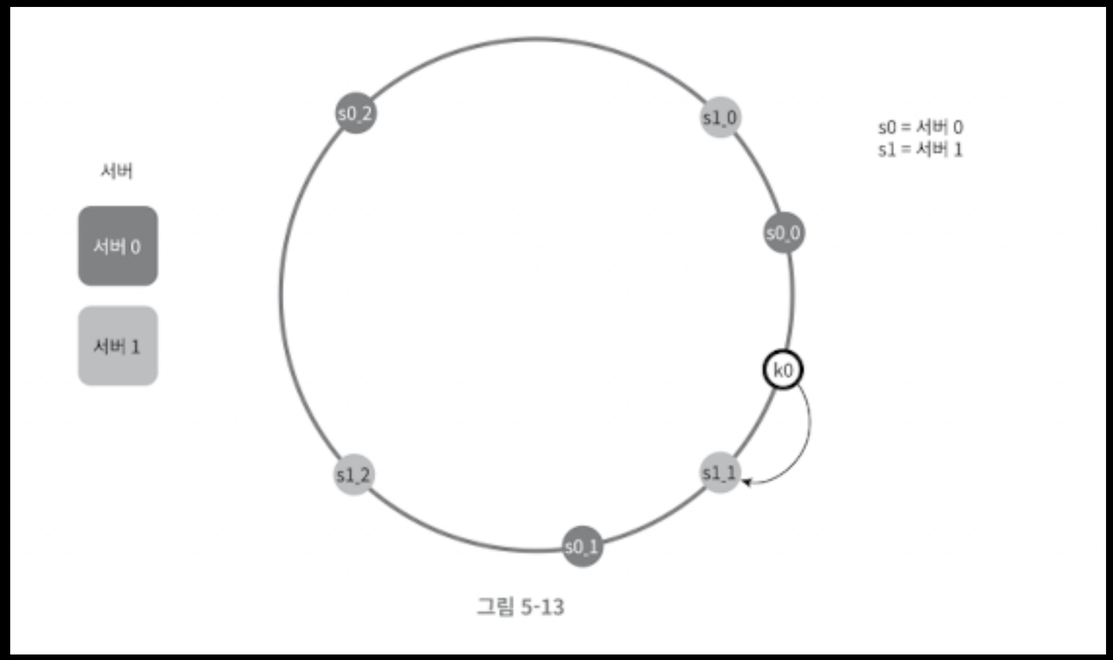


그림 5-13

2. 안정 해시

- 재배치할 키 결정

- 서버가 추가되거나 제거되면, 데이터의 일부는 재배치해야 함

- 서버 4가 추가될 경우, 영향을 받는 것은 해당 직전 서버인 s3부터 s4까지의 키들임

- s3부터 s4 사이에 있는 키들을 s4로 재배치해야 함

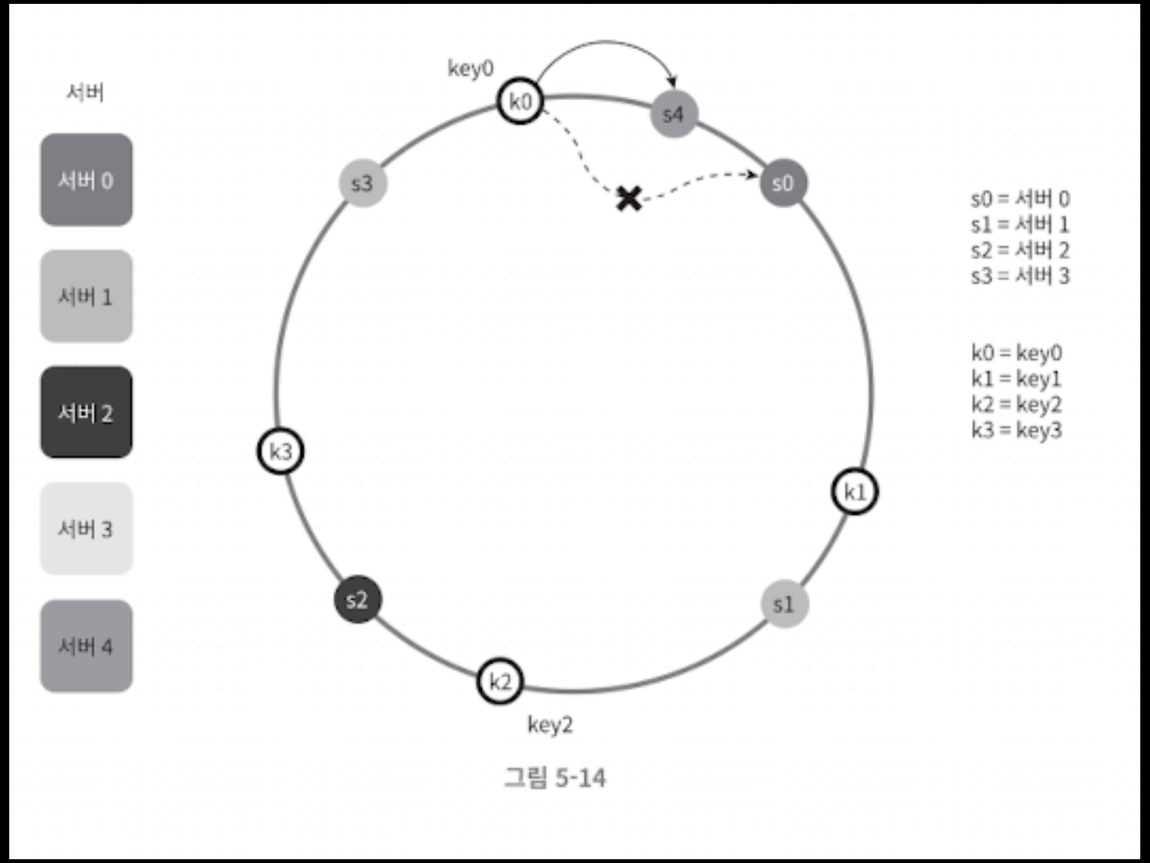


그림 5-14

3. 참고자료