[5장] 안정 해시 설계

가상 면접 사례로 배우는 대규모 시스템 설계 기초

이민석 / unchaptered

https://inblog.ai/monthly-cs
https://github.com/monthly-cs/2024-03-system-design-interview-1

해시(Hash)란?

입력값(Input)을 **고정된 길이의 값(Output)**으로 변환(매핑)하는 함수

```go
func main() {

    keys := []string{
        "key1",
        "key2",
        "key3",
        "key4",
        "key5",
        "key6",
        "key7",
        "key8",
    }
    N := 10

    for i := range keys {
        servcerIndex := hash(keys[i]) % uint32(N)
        fmt.Printf("Index\t[%d]\t%s\t%d\t%d\n", i, keys[i], hash(keys[i]), servcerIndex)
    }
}
```

```
C:\Personal\learn-for-golang\src\hash>go run main
Index   [0]     key1    927623783       3
Index   [1]     key2    944401402       2
Index   [2]     key3    961179021       1
Index   [3]     key4    843735688       8
Index   [4]     key5    860513307       7
Index   [5]     key6    877290926       6
Index   [6]     key7    894068545       5
Index   [7]     key8    776625212       2
```

Input Output

https://github.com/unchaptered/learn-for-golang/blob/main/src/hash/main.go

해시(Hash)의 특징

32비트 해시 함수를 사용했으며, 2^32개의 서로 다른 해시값을 생성 가능
해시 함수는 가능한 입력값(Input)을 골고루 해시값(Output)으로 분산시켜야 함

입력값(변수명:keys)이 많아지면 해시 충돌이 날 가능성이 높아짐

```go
func hash(s string) uint32 {
    h := fnv.New32a()
    h.Write([]byte(s))
    return h.Sum32()
}

func main() {

    keys := []string{···
    }
    N := 10

    for i := range keys {
        servcerIndex := hash(keys[i]) % uint32(N)
        fmt.Printf("Index\t[%d]\t%s\t%d\t%d\n", i, keys[i], hash(keys[i]), servcerIndex)
    }
}
```

https://github.com/unchaptered/learn-for-golang/blob/main/src/hash/main.go

해시(Hash)를 활용한 서버 분산

각 해시값(Output)을 서버의 수(N)로 **나머지 연산**을 하여, **서버의 인덱스**를 계산

- ServerIndex = Hash(Key) % N

```go
func hash(s string) uint32 {
    h := fnv.New32a()
    h.Write([]byte(s))
    return h.Sum32()
}

func main() {

    keys := []string{···
    }
    N := 10

    for i := range keys {
        servcerIndex := hash(keys[i]) % uint32(N)
        fmt.Printf("Index\t[%d]\t%s\t%d\t%d\n", i, keys[i], hash(keys[i]), servcerIndex)
    }
}
```

```
C:\Personal\learn-for-golang\src\hash>go run main
Index   [0]     key1    927623783       3
Index   [1]     key2    944401402       2
Index   [2]     key3    961179021       1
Index   [3]     key4    843735688       8
Index   [4]     key5    860513307       7
Index   [5]     key6    877290926       6
Index   [6]     key7    894068545       5
Index   [7]     key8    776625212       2
```

https://github.com/unchaptered/learn-for-golang/blob/main/src/hash/main.go

해시(Hash)의 문제점

32비트 해시 함수의 경우 2^32 만큼의 경우의 수가 있음
하지만 입력값의 전달 유형에 따라서 **낮은 분포도**를 보일 수 있다.

또한 앞서 언급한 **ServerIndex**가 특정 구간에 밀집될 확률이 존재합니다.
기본 해시함수의 경우, 이 ServerIndex가 특정 구간에 밀집될 확률이 존재합니다.

```
C:\Personal\learn-for-golang\src\hash>go run main
Index   [0]     key1    927623783       3
Index   [1]     key2    944401402       2
Index   [2]     key3    961179021       1
Index   [3]     key4    843735688       8
Index   [4]     key5    860513307       7
Index   [5]     key6    877290926       6
Index   [6]     key7    894068545       5
Index   [7]     key8    776625212       2
```

https://github.com/unchaptered/learn-for-golang/blob/main/src/hash/main.go

해시(Hash)의 재배열

만약 장애가 발생하여 **서버의 수(N)**가 1개 줄어들면, 모든 서버의 ServerIndex가 달라지게 된다. 따라서, 모든 요청이 **기존과 다른 서버**로 향하게 되고 대량의 CacheMiss가 발생

```
C:\Personal\learn-for-golang\src\hash>go run main
Index   [0]     key1    927623783       3
Index   [1]     key2    944401402       2
Index   [2]     key3    961179021       1
Index   [3]     key4    843735688       8
Index   [4]     key5    860513307       7
Index   [5]     key6    877290926       6
Index   [6]     key7    894068545       5
Index   [7]     key8    776625212       2
```

https://github.com/unchaptered/learn-for-golang/blob/main/src/hash/main.go

안정 해시(Consistence Hash)란?

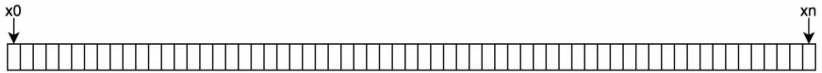
해시 테이블 크기(N)가 조정될 때, 오직 해당되는 키(k/n) 개의 키만 재배치되는 해시 기술

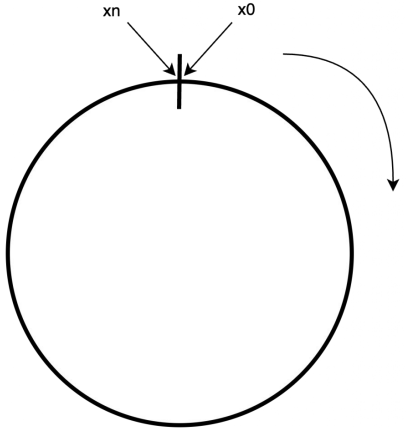
- 키가 10개고 서버가 5개인 상황에서, 서버가 1개 줄어들면 10/5 개의 키만 재배치

해시 공간, 링

해시 알고리즘에 따라서 사용가능한 공간을 리스트(List) 형태로 표현한 것을 **해시 공간**으로

이의 양쪽 끝을 연결하여 순환 리스트(Circular List) 형태로 표현한것을 **해시 링**으로 표기

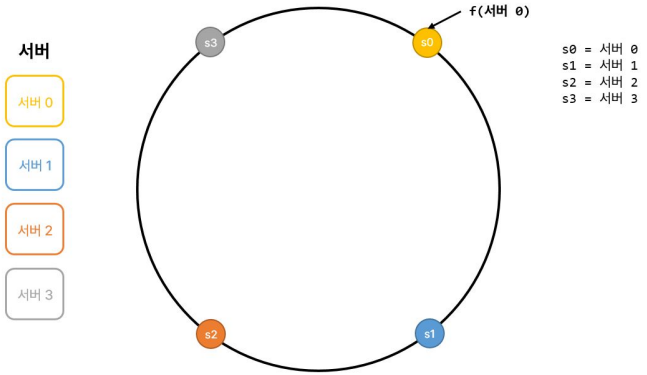




https://jonghoonpark.com/2023/05/25/%EC%95%88%EC%A0%95-%ED%95%B4%EC%8B%9C-%EC%84%A4%EA%B3%84

해시 서버

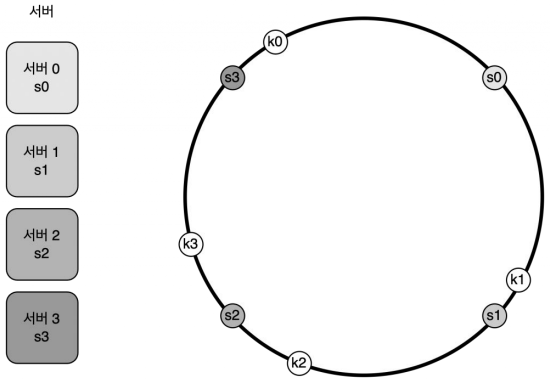
해시 **함수 f**를 사용하여 **서버 IP나 이름(Sn)**을 해시 링에 대응시킬 수 있음



https://velog.io/@mmy789/System-Design-6

해시 키

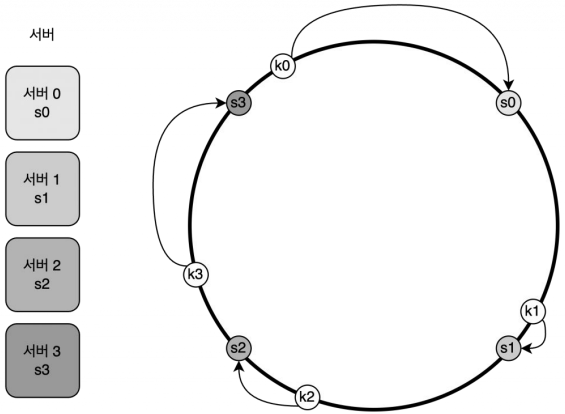
캐시할 키(Kn)들도 해시 링 위의 어느 지점으로 배치할 수 있음



https://jonghoonpark.com/2023/05/25/%EC%95%88%EC%A0%95-%ED%95%B4%EC%8B%9C-%EC%84%A4%EA%B3%84

서버 조회

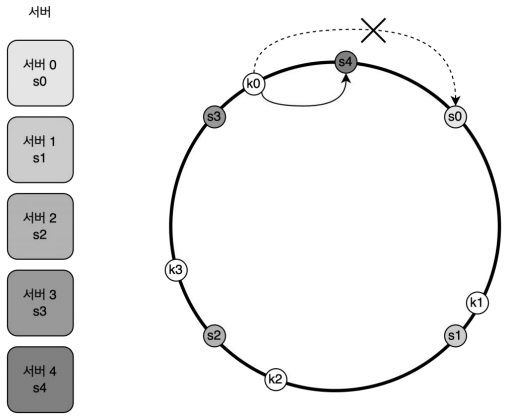
어떤 키(Kn)이 어떤 서버(Sn)로 요청을 보내는 지는 해시 링을 **시계 방향으로 탐색**해서 결정



https://jonghoonpark.com/2023/05/25/%EC%95%88%EC%A0%95-%ED%95%B4%EC%8B%9C-%EC%84%A4%EA%B3%84

서버 추가

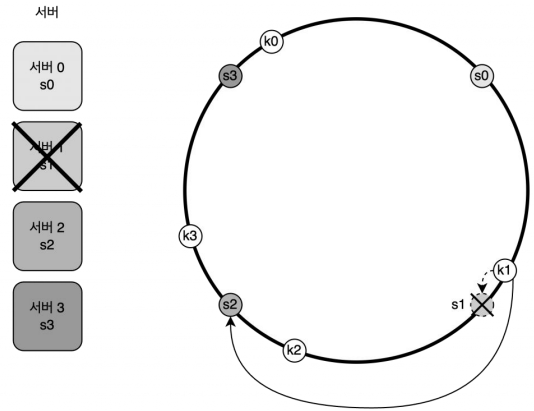
기존의 키(K0)과 기존 서버(S0) 사이에 신규 서버(S4)가 추가되면,
기존의 키(K0)의 요청이 신규 서버로 변경될 수 있다.



https://jonghoonpark.com/2023/05/25/%EC%95%88%EC%A0%95-%ED%95%B4%EC%8B%9C-%EC%84%A4%EA%B3%84

서버 제거

한 키(K1)와 한 서버(S1)가 있을 때,
한 서버(S1)가 제거되면 **해시 링의 다음 번의 나오는 서버**로 변경된다.

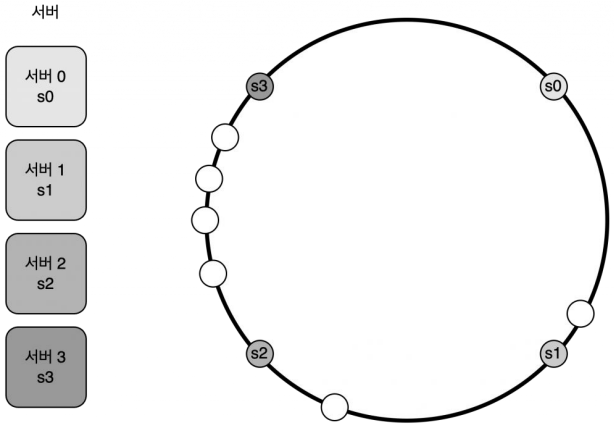


https://jonghoonpark.com/2023/05/25/%EC%95%88%EC%A0%95-%ED%95%B4%EC%8B%9C-%EC%84%A4%EA%B3%84

기본 구현 법의 두 가지 문제

서버의 추가, 제거 과정에 따라서 특정한 서버(Sx)에 대량의 키(Ka, Kb, Kc)가 몰릴 수 있음

1. 서버 파티션의 크기를 균등하게 유지할 수 없음
2. 키의 균등 분포를 담당할 수 없음

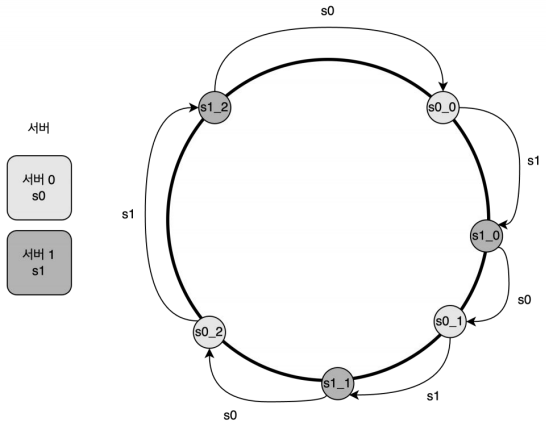


https://jonghoonpark.com/2023/05/25/%EC%95%88%EC%A0%95-%ED%95%B4%EC%8B%9C-%EC%84%A4%EA%B3%84

가상 노드

가상 노드는 실제 서버를 가리키는 노드

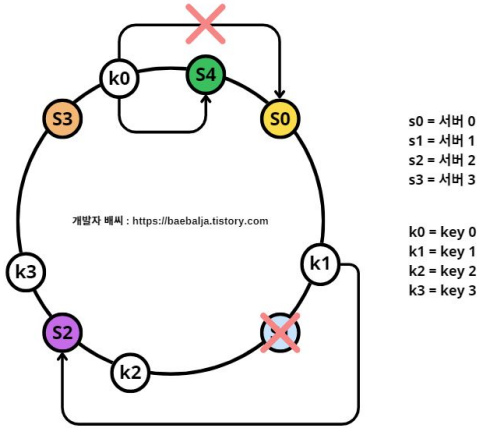
특정 키(Kn)가 실제 서버(Sn)을 가리키도록 하는 것이 아니라, 실제 서버를 가리키고 있는 가상 노드(Sn_n)을 가리키도록 함



https://jonghoonpark.com/2023/05/25/%EC%95%88%EC%A0%95-%ED%95%B4%EC%8B%9C-%EC%84%A4%EA%B3%84

재배치 결정

해서는 요청 또는 데이터를 서버에 균등하게 나누는 것이 중요



https://baebalja.tistory.com/588

결론

- 임의의 요청을 균등하게 분배하는데 해쉬(Hash)를 사용할 수 있다.
- 해쉬는 여러 가지 문제를 가지고 있어 안정해쉬를 써야한다.
- 안정해쉬를 구현할 때에는 가상 노드를 사용하여 분포를 고르게 유지해야 한다.

감사합니다.