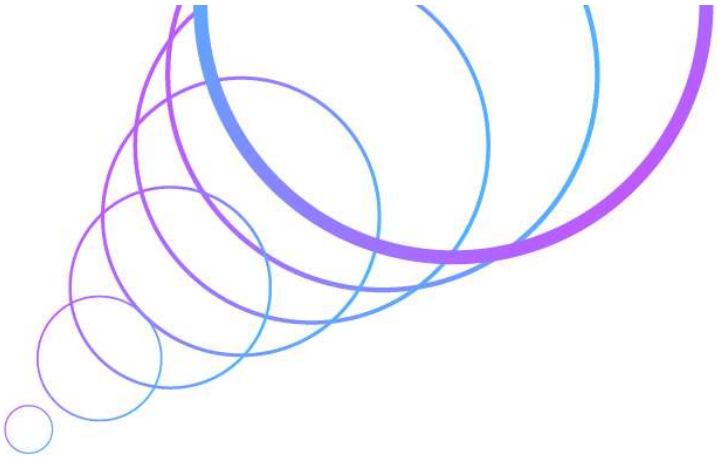


글로벌 데이터 주권 강화 시대에 맞는 데이터 CUBE 개발 (GDPR 에 맞서는 개발)

NAVER / Log-Data : 김영진, 이모원

CONTENTS

1. Backgrounds and issues

1.1 “데이터 주권 주의” 란

DEVIEW 2021

데이터 주권 주의란.

- 사람의 신체와 같이 데이터의 권리 또한 사용자 개개인에게 있다는 개념.
- 국가 차원에서의 자국 데이터 산업 보호를 위한 개념. (중국, 미국, 러시아, 일본 등등)

네이버 스마트스토어, 웹툰 등 세계로 뻗어 나가는 서비스 등장

⇨ 다양한 형태의 대용량 데이터 분석 요구가 증가

"데이터 이동을 제한하는 각 국가별 관련 법규를 준수하는 것은 선택이 아닌 필수”

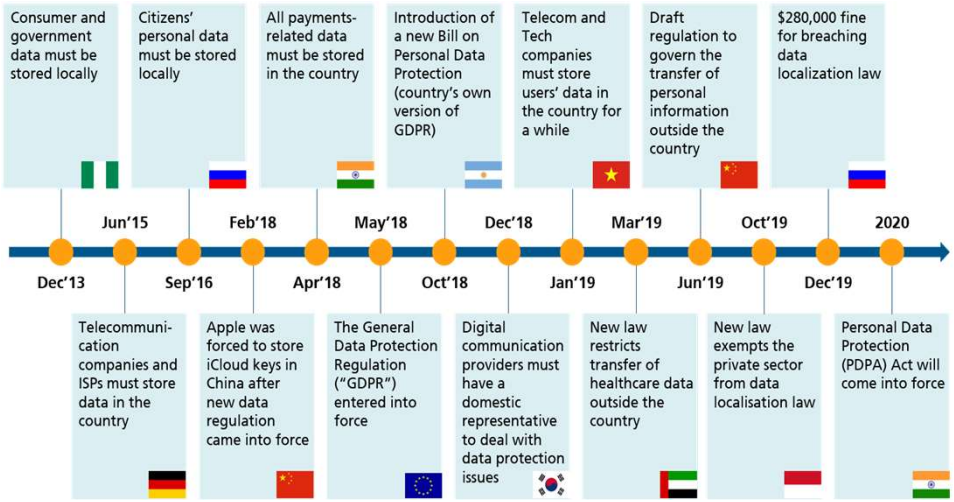
➔ 유연한 데이터 분석 기술 개발이 필요한 시점

1.2 “데이터 이동 제한” 이란

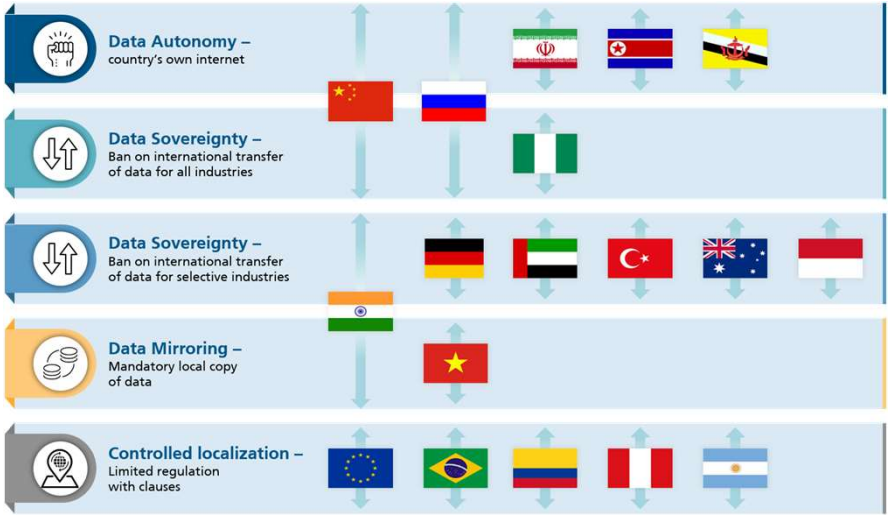
DEVIEW 2021

데이터 이동 제한이란 (Data Localization)

- 수집된 사용자의 데이터를 타국으로 반출하는 것을 제한 (예: GDPR 등)
- 국가 마다 데이터 이동 제한 정책이 매우 다름



국가별 데이터 정책 수립 Timeline



다양한 국가별 데이터 지역 정책

https://www.deltapartnersgroup.com/data-localisation-information-protection-balkanisation-internet

1.3 “데이터 이동 제한” 지키지 않는다면?

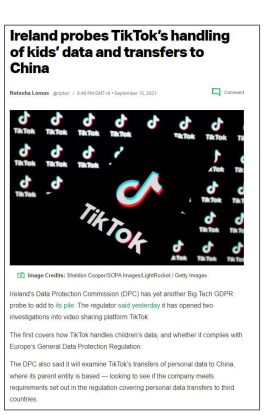
DEVIEW 2021

금전적인 손해
국가와 회사에 대한 이미지 실추

위반 사례)
- facebook 개인정보 위탁 및 국외 이전 관련 내용 미공개 등으로 과태로 2600만원
- Google 국외 이전 개인정보 항목의 구체적 명시 등으로 인한 개선 권고
- 등등

https://m.etnews.com/20210825000176

국가별로 제제가 계속 되고 있는 사례)
- 아일랜드 DPC 의 TikTok 의 사용자 데이터 중국 이전 조사
- 중국의 PIPL 위반시 최대 90억원
- 국내도 EU GDPR과 동등한 수준으로 법체계 마련

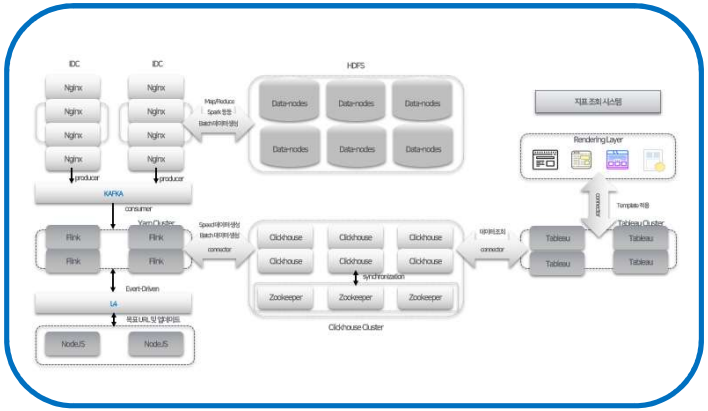


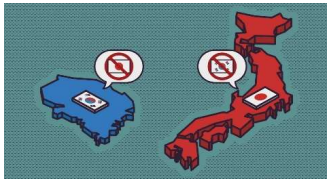
https://techcrunch.com/2021/09/15/ireland-probes-tiktoks-handling-of-kids-data-and-transfers-to-china/
http://www.mk.co.kr/news/world/view/2021/08/821075/
https://m.etnews.com/20210330000245

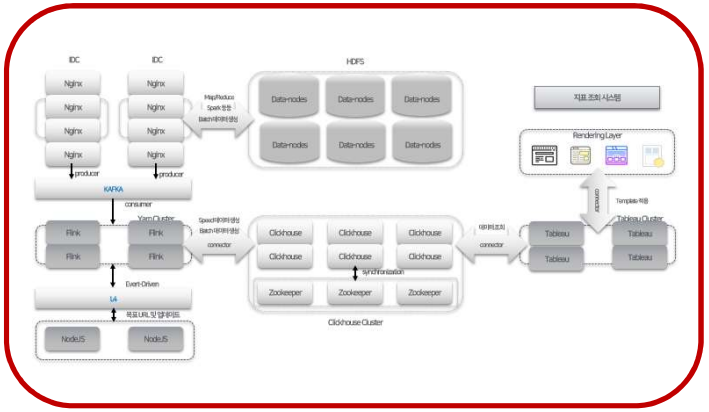
1.4 피할 수 없는 네이버 로그의 파편화

DEVIEW 2021

로그수집에서 지표 조회까지 모든 데이터 처리/저장은 로그 수집이 있던 국가의 IDC 에서 진행



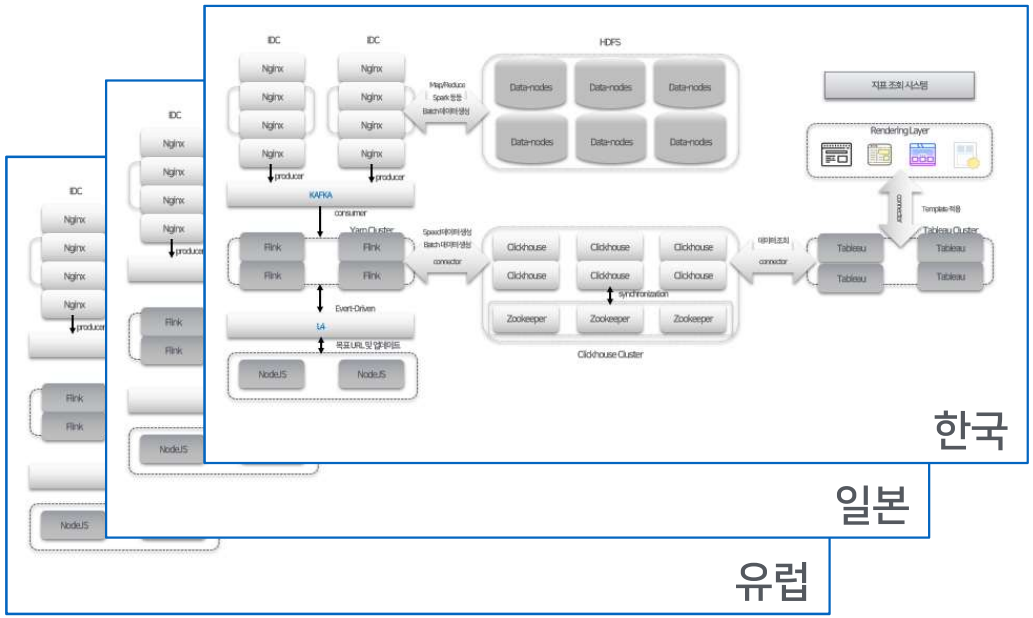




- B/E, F/E 등 동일한 시스템을 복수개로 유지
- 유니크 계산이 불가능하거나 굉장히 불필요하고 비효율적인 작업들이 발생

이미지: http://tongsangnews.kr/webzine/2107/sub1_1.html

1.5 데이터 시스템 분리 구축 시 발생되는 문제



• • •

회사 입장

- infra 비용 발생
- 불필요한 현지인력 채용 증가

개발자 입장

- 굉장히 하기 싫은 업무로 전락
- 유지보수 증가

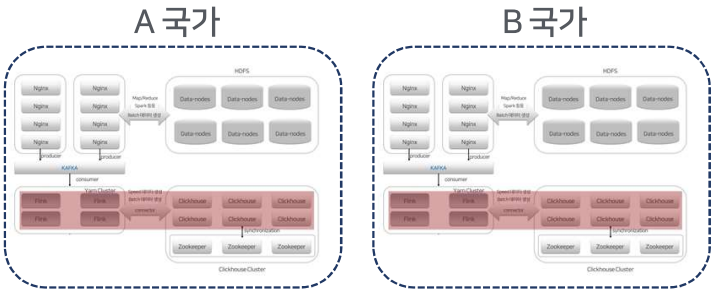
1.6 절대 할 수 없는 것과 도전할 수 있는 부분

DEVIEW 2021

법규 준수를 위해 피할 수 없는 부분

로그 수집 & 로그 저장 인프라

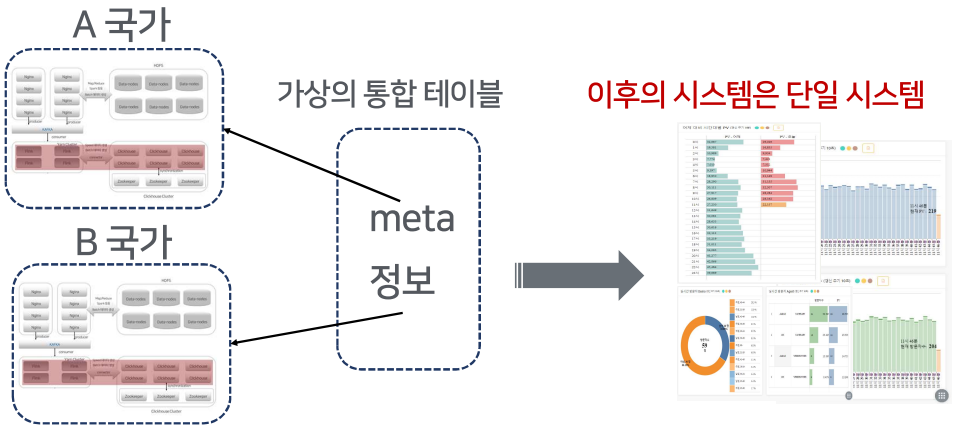
지표값 이외의 정보가 담긴 중간 집계 데이터



도전 할 수 있는 부분

가상의 통합 테이블 구축

집계 & 저장 이후의 프로세스는 단일화



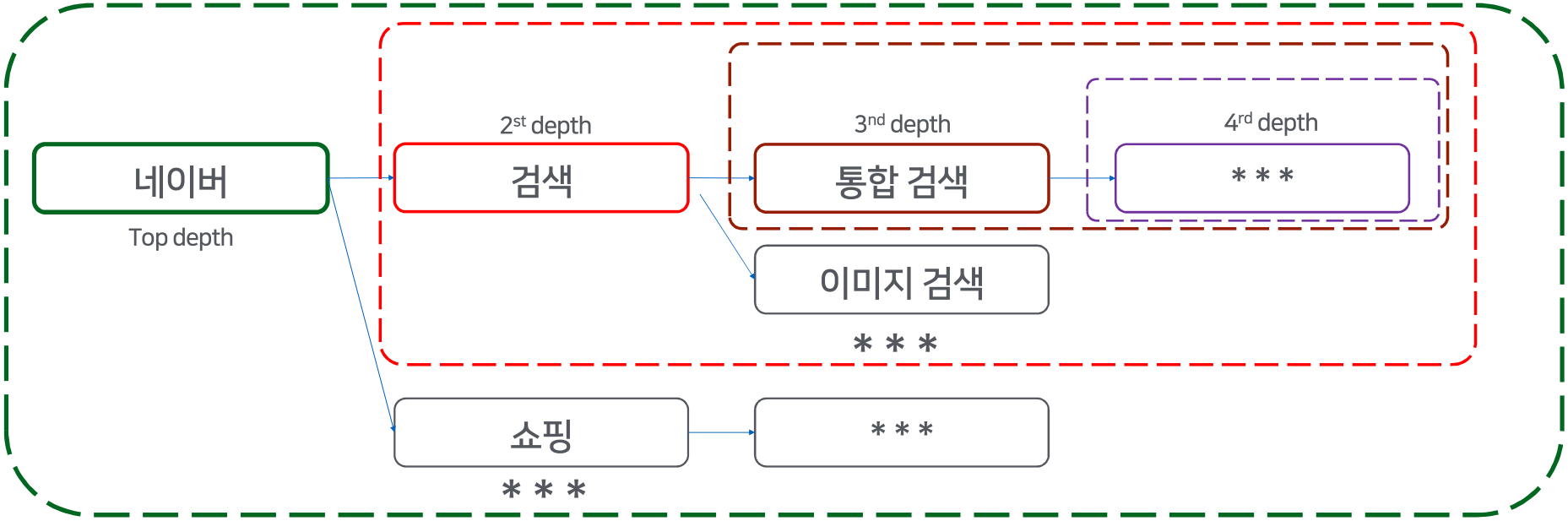
실시간 지표 등을 고려해서 고성능 READ 필요

2. 네이버 지표의 특성

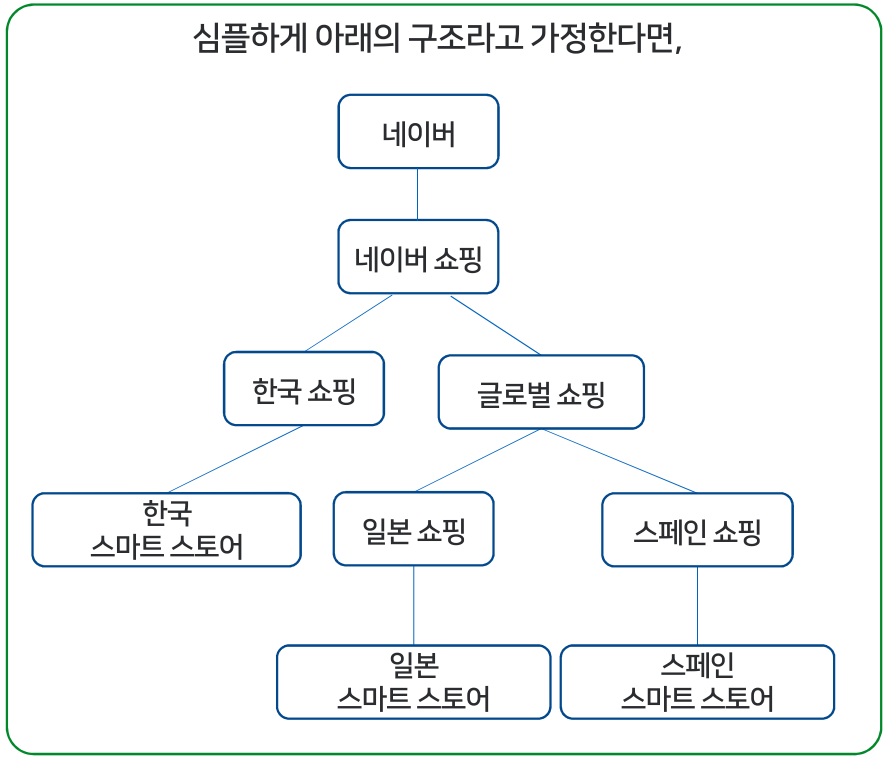
2.1 네이버 서비스의 구조적인 특징

네이버는 포털 서비스

- 검색, 쇼핑, 페이, 뉴스 등등 다양한 서비스를 포함하는 포털이라는 개념
- 개별 서비스는 네이버 메인과 검색 서비스를 통한 유입이 많은 특징
- 개별 서비스도 중요하겠지만 거시적인 관점에서의 뷰가 중요



2.2 네이버 서비스의 유연한 지표 특징



ex) 일본스마트스토어의 서비스키

네이버/네이버쇼핑/글로벌쇼핑/일본쇼핑/일본스마트스토어

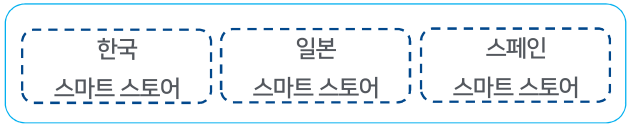
구조적 지표 관점

서비스 key를 tree 형태의 구조로 잡혀 있기 때문에

하위 노드의 합이나 유니크 계산을 자연스럽게 할 수 있는 구조

커스텀 지표 관점

Tree의 방향도 다르고, Tree의 depth도 다른 NODE 들의 통합지표 제공



의 통합 pv와 user수 등의 지표

PV는 합산으로 문제해결, 하지만 user수는... ?

2.3 네이버 서비스의 유연한 지표 제공을 위한 난관

DEVIEW 2021

심플하게 로그가 아래와 같이 있다면,

Row#	date	service	Visitor	PageView
1	2021-08-20 09:10:10	/네이버/네이버쇼핑/한국쇼핑/한국스마트스토어	사용자1, 사용자3, 사용자5	23
2	2021-08-20 09:10:10	/네이버/네이버쇼핑/글로벌쇼핑/일본쇼핑/일본스마트스토어	사용자2, 사용자3, 사용자5	24
3	2021-08-20 09:10:10	/네이버/네이버쇼핑/글로벌쇼핑/스페인쇼핑/스페인스마트스토어	사용자4, 사용자 5	25

PV 집계

한국스마트스토어 23 + 일본스마트스토어 24 + 스페인스마트스토어 25

➔ 72

각 IDC에서 집계된 값을 단순 SUM 해서 해결이 가능

user수 집계

한국스마트스토어 3 + 일본스마트스토어 3 + 스페인스마트스토어 2

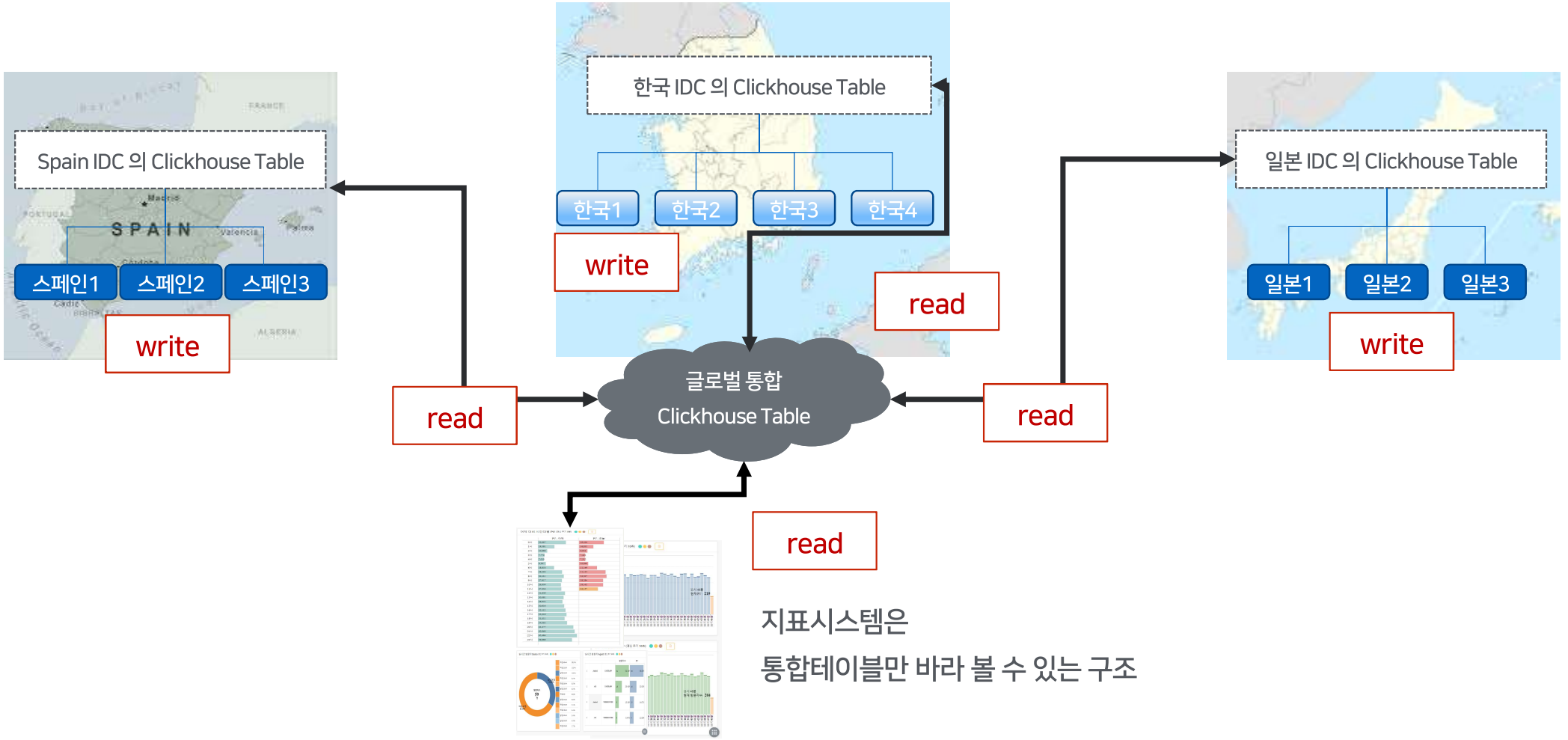
➔ 8

각 IDC에서 집계된 값을 단순 SUM 해서 해결이 불가능

uniq (a) + unique(b) != uniq (a + b)

3.글로벌 통합 가상 테이블 구축

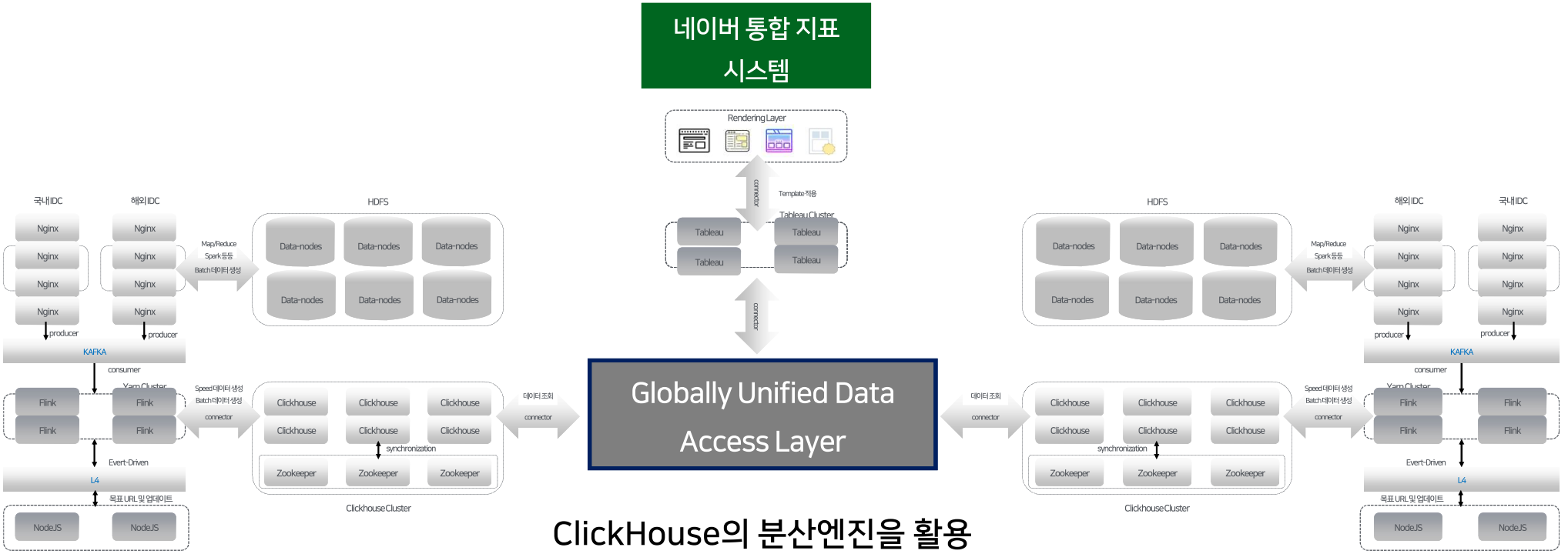
3.1 글로벌 통합 가상 테이블



지표시스템은

통합테이블만 바라 볼 수 있는 구조

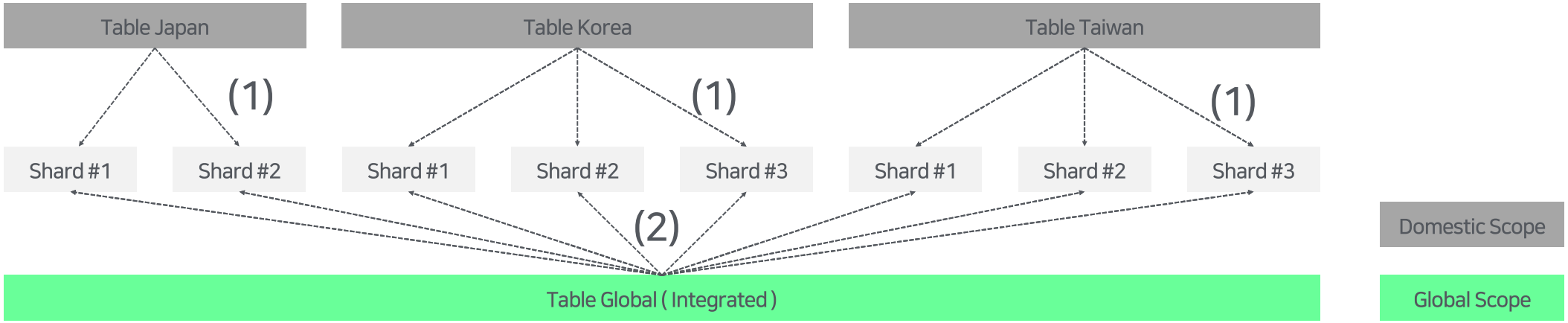
3.2 ClickHouse 분산 테이블 엔진을 활용한 분리 구축



ClickHouse의 분산엔진을 활용

4. 글로벌 Clickhouse 구성

4.1 글로벌 Clickhouse 구성 (예)



MergeTree Table Engine 기반

- LSM Tree 기반의 table engine
- 이 자체로 Index 가 존재하며 독립적인 Table Component

Distributed Table Engine 기반

- Persistent Data 저장소는 아님 (Write 시 MergeTree Engine 으로 가기 위한 Buffer)
- Write 시 Data 를 분리하는 Shard Key를 정의하고
- Read시 연결된 MergeTree Engine의 Data 를 취합 (Requestor/Remote Node)

4.2 글로벌 Clickhouse 구성 방법 - Shard

DEVIEW 2021

Clickhouse Table Configuration - Shard

```
CREATE TABLE test
(
    id Int64,
    partition Int16
)
ENGINE = MergeTree()
ORDER BY id
PARTITION BY partition;
```

(1)

Table creation – 1,2 번 노드의 MergeTree Table 생성

```
CREATE TABLE distributed_test ON CLUSTER
(
    id Int64,
    partition Int16
)
ENGINE = Distributed ( '2shard_cluster', '', test, sipHash( partition ) )
ORDER BY id
PARTITION BY partition;
```

(2)

Table 생성 – 1,2, 번 노드를 아우르는 Distribution Table 생성 (동일하게 1,2 번 노드에서 생성)

```
<2shard_cluster> 설정에서 사전 정의되어야 할 Cluster 구성
        <shard>
            <replica>
                <host>node1</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <replica>
                <host> node1 </host>
                <port>9000</port>
            </replica>
        </shard>
</2shard_cluster>
```

(3)

Configuration

4.3 글로벌 Clickhouse 구성 방법 - Replication

DEVIEW 2021

Clickhouse Table Configuration - Replication

Table creation on 1st database

```
CREATE TABLE test
(
    id Int64,
    partition Int16
)
ENGINE = ReplicatedMergeTree('/Clickhouse/tables/replicated/test', 'replica_1')
ORDER BY id
PARTITION BY partition;
```

동일한 Zookeeper path

Replica Name

Table creation on 2nd database

```
CREATE TABLE test
(
    id Int64,
    partition Int16
)
ENGINE = ReplicatedMergeTree('/Clickhouse/tables/replicated/test', 'replica_2')
ORDER BY id
PARTITION BY partition;
```

always_fetch_merged_part

Prohibits data parts merging in Replicated*MergeTree-engine tables.

When merging is prohibited, the replica never merges parts and always downloads merged parts from other replicas. If there is no required data yet, the replica waits for it. CPU and disk load on the replica server decreases, but the network load on the cluster increases. This setting can be useful on servers with relatively weak CPUs or slow disks, such as servers for backups storage.

Possible values:

- 0 — `Replicated*MergeTree`-engine tables merge data parts at the replica.
- 1 — `Replicated*MergeTree`-engine tables do not merge data parts at the replica. The tables download merged data parts from other replicas.

Default value: 0.

```
<shard>
 <replica>
  <default_database>r0</default_database>
   <host>ach011.nds.nfra.io</host>
   <port>9000</port>
 </replica>
 <replica>
  <default_database>r1</default_database>
   <host>ach012.nds.nfra.io</host>
   <port>9000</port>
 </replica>
 <internal_replication>true</internal_replication>
</shard>
```

Configuration

4.4 글로벌 Clickhouse 구성 – for Write

DEVIEW 2021

Clickhouse Table Configuration – Shard & Replication

#1

```
CREATE TABLE r0.lcs_url_array_basic
(
        date Date,
        URL String,
        pv UInt64
) ENGINE =
ReplicatedMergeTree('/Clickhouse/cluster_202101/lcs_url_array_basic_0', '0')
PARTITION BY toYYYYMM( date )
ORDER BY ( URL )
```

#2

```
CREATE TABLE r0.lcs_url_array_basic
(
        date Date,
        URL String,
        pv UInt64
) ENGINE =
ReplicatedMergeTree('/Clickhouse/cluster_202101/lcs_url_array_basic_1', '1')
PARTITION BY toYYYYMM( date )
ORDER BY ( URL )
```

#3

```
CREATE TABLE r0.lcs_url_array_basic
(
        date Date,
        URL String,
        pv UInt64
) ENGINE =
ReplicatedMergeTree('/Clickhouse/cluster_202101/lcs_url_array_basic_2', '1')
PARTITION BY toYYYYMM( date )
ORDER BY ( URL )
```

```
CREATE TABLE default.dlcs_url_array_basic
(
        date Date,
        URL String,
        pv UInt64
) ENGINE=Distributed( 'cluster_202101', '', lcs_url_array_basic, sipHash64( URL ))
```

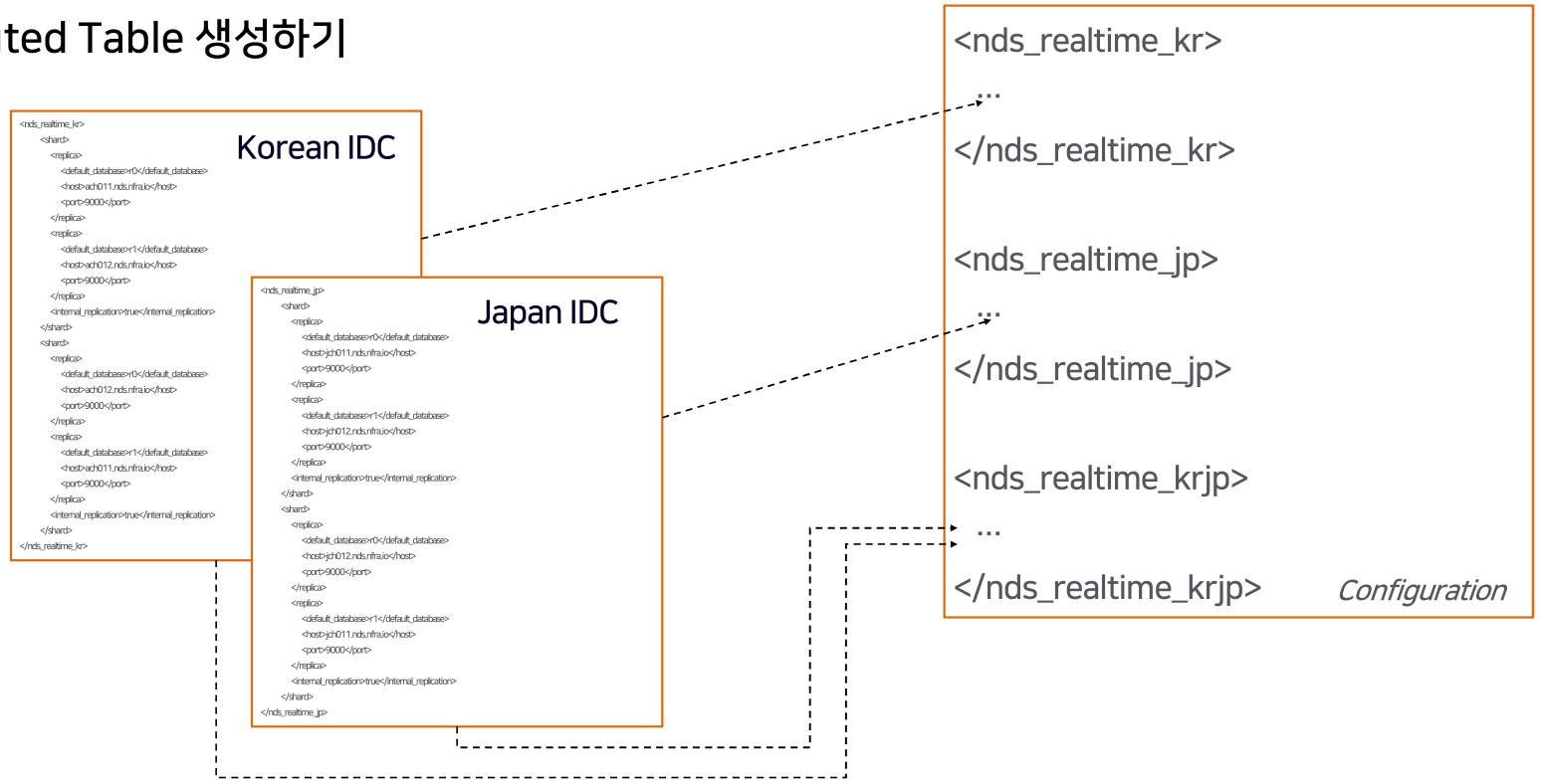
Configuration

```
<shard>
    <replica>
        <default_database>r0</default_database>
        <host>ach011.nds.nfra.io</host>
        <port>9000</port>
    </replica>
    <replica>
        <default_database>r1</default_database>
        <host>ach012.nds.nfra.io</host>
        <port>9000</port>
    </replica>
    <internal_replication>true</internal_replication>
</shard>
 ……
<shard>
    <replica>
        <default_database>r0</default_database>
        <host>ach013.nds.nfra.io</host>
        <port>9000</port>
    </replica>
    <replica>
        <default_database>r1</default_database>
        <host>ach011.nds.nfra.io</host>
        <port>9000</port>
    </replica>
    <internal_replication>true</internal_replication>
</shard>
```

4.5 글로벌 Clickhouse 구성 – for Read

DEVIEW 2021

통합 Distributed Table 생성하기



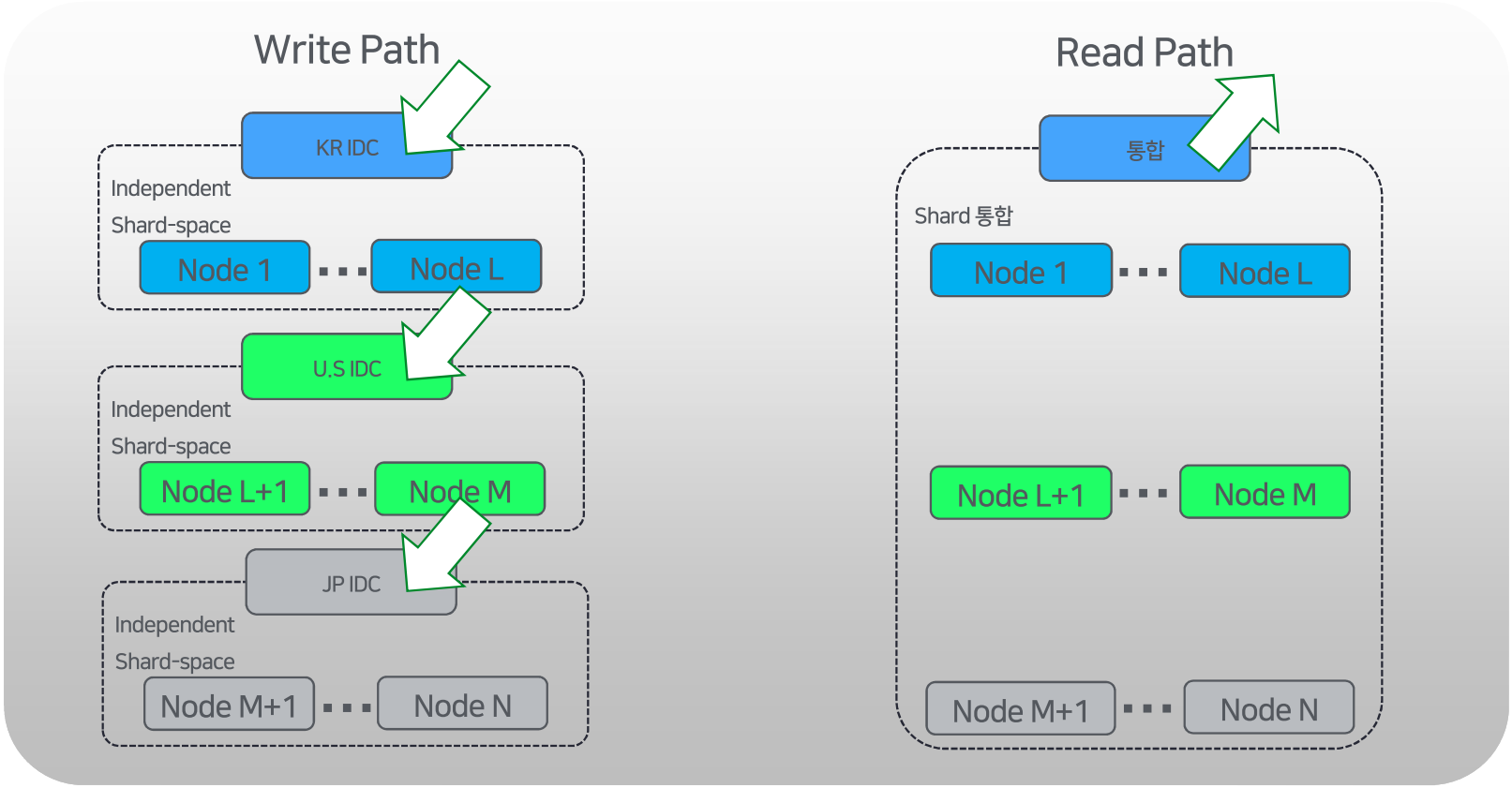
한국

```
CREATE TABLE default.dlcs_url_array_basic ( date Date, URL String, pv UInt64 ) ENGINE=Distributed( nds_realtime_krjp', '', lcs_url_array_basic, sipHash64( URL ))
CREATE TABLE default.dlcs_url_array_basic ( date Date, URL String, pv UInt64 ) ENGINE=Distributed( nds_realtime_jp', '', lcs_url_array_basic, sipHash64( URL ))
CREATE TABLE default.dlcs_url_array_basic_krjp ( date Date, URL String, pv UInt64 ) ENGINE=Distributed( nds_realtime_krjp', '', lcs_url_array_basic, sipHash64( URL ))
```

일본

4.6 글로벌 Clickhouse 구성

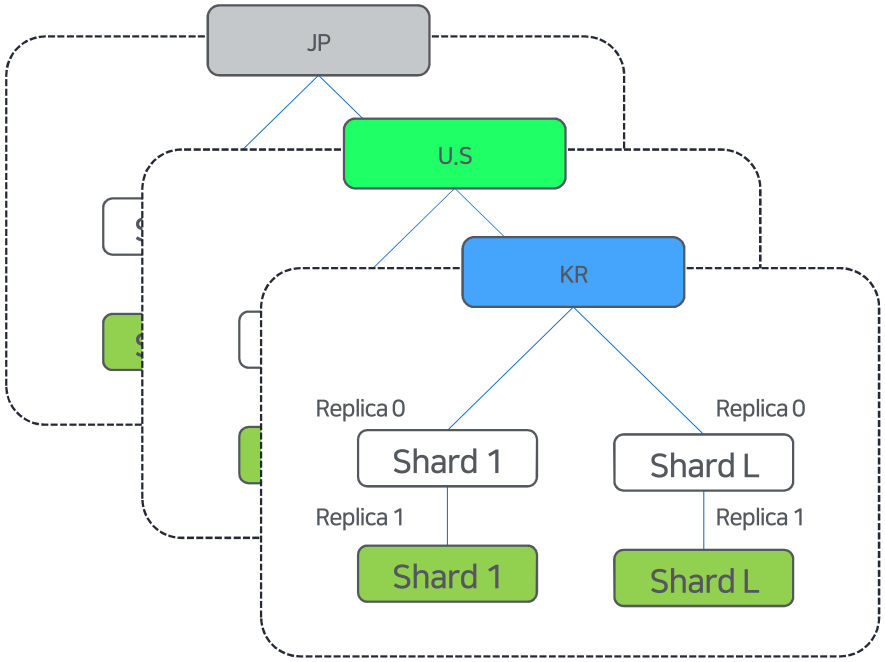
중첩된 Distributed Table 생성을 통해 Read/Write Path 를 분리한 것이 우리의 Key



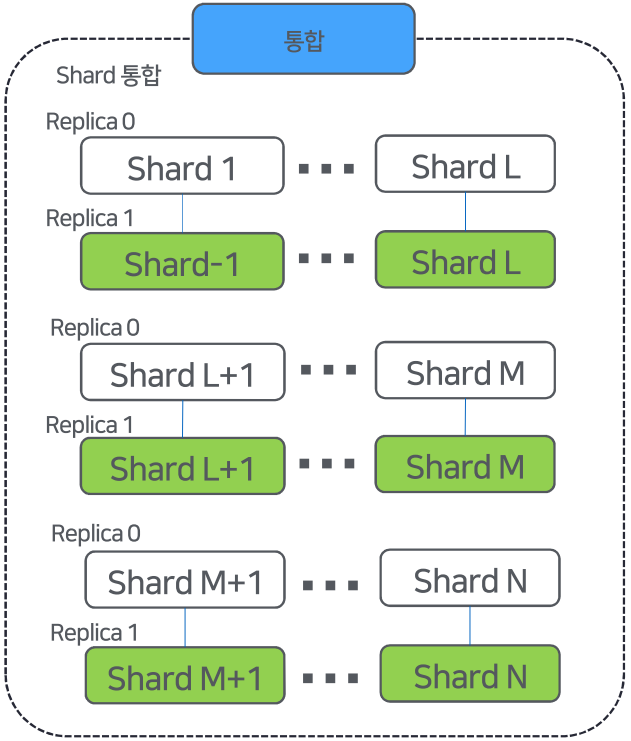
4.6 글로벌 Clickhouse 구성

또한 기존과 동일 H/A topology 를 유지 할 수 있다.

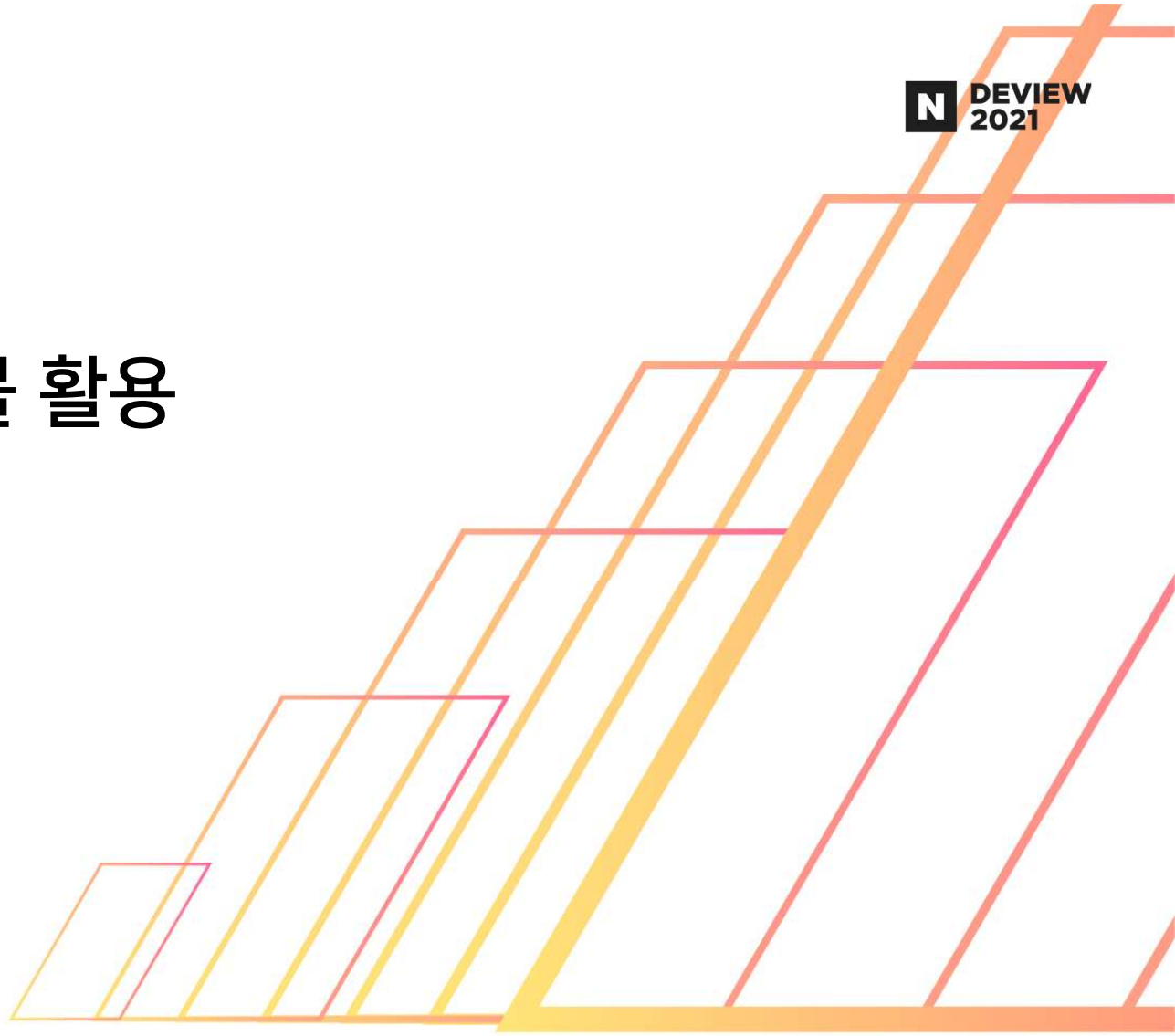
Write Path



Read Path



5. 통합 테이블 활용



5.1 통합 테이블 활용 case 1

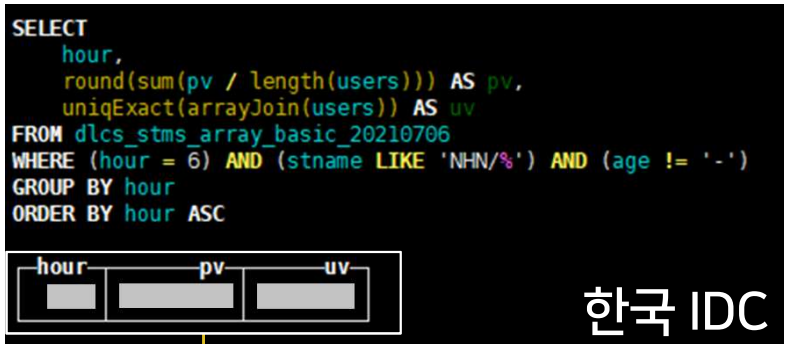
DEVIEW 2021

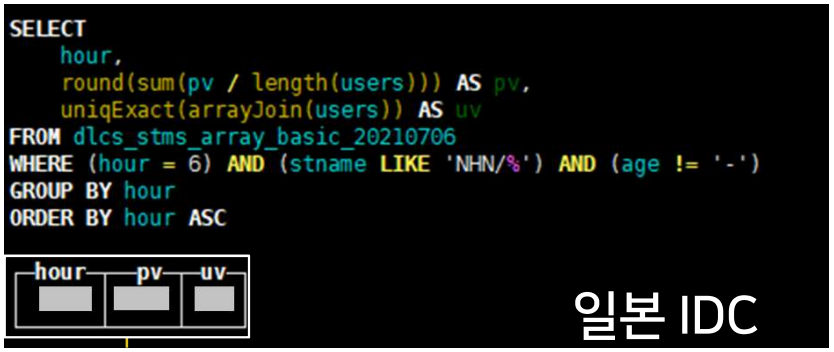
Clickhouse 분산 테이블 기반의 Unique 사용자 계산 (TOP 노드 이하 모든 데이터)

Table Japan | Table Korea | Table Taiwan

```
SELECT
    hour,
    round(sum(pv / length(users))) AS pv,
    uniq(arrayJoin(users)) AS uv
FROM dlcs_stms_array_basic_20210706
WHERE (stname LIKE 'NHN/%') AND (age != '-')
GROUP BY hour
ORDER BY hour ASC
```

합산





5.2 통합 테이블 활용 case 2

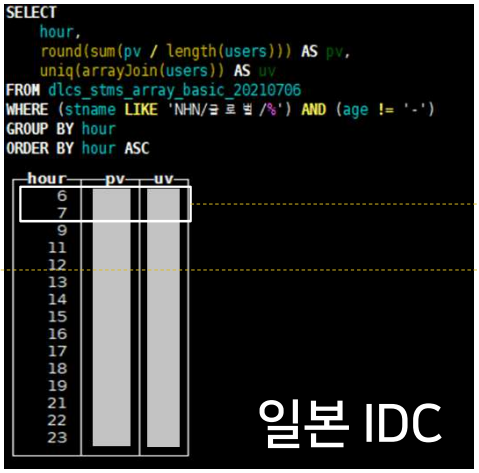
DEVIEW 2021

Clickhouse 분산 테이블 기반의 Unique 사용자 계산 (글로벌 IDC 데이터 대상)

```
SELECT
    hour,
    round(sum(pv / length(users))) AS pv,
    uniq(arrayJoin(users)) AS uv
FROM dlcs_stms_array_basic_20210706
WHERE (stname LIKE 'NHN/글로벌/%') AND (age != '-')
GROUP BY hour
ORDER BY hour ASC
```

Table Japan

Table Taiwan



합산

5.3 통합 테이블 활용 case 3

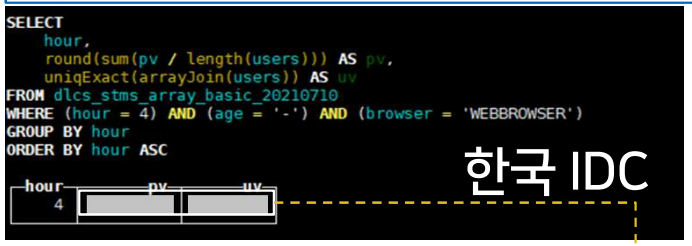
DEVIEW 2021

모든 IDC 에 걸쳐 WEBBROWSER 사용자의 PV 와 UV 계산

```
SELECT
    hour,
    round(sum(pv / length(users))) AS pv,
    uniq(arrayJoin(users)) AS uv
FROM dlcs_stms_array_basic_krjp_20210710
WHERE (age = '-') AND (browser = 'WEBBROWSER')
GROUP BY hour ORDER BY hour ASC
```

Table Japan | Table Korea | Table Taiwan

(모든 IDC 에서 Service Tree 와 상관 없이)



6. 왜 Clickhouse 였는가?

6.1 LSM-Tree 를 통해서 본 Clickhouse 의 특징

DEVIEW 2021

모든 데이터는 Append 형태로 Write된다. (빠른 Write 성능)

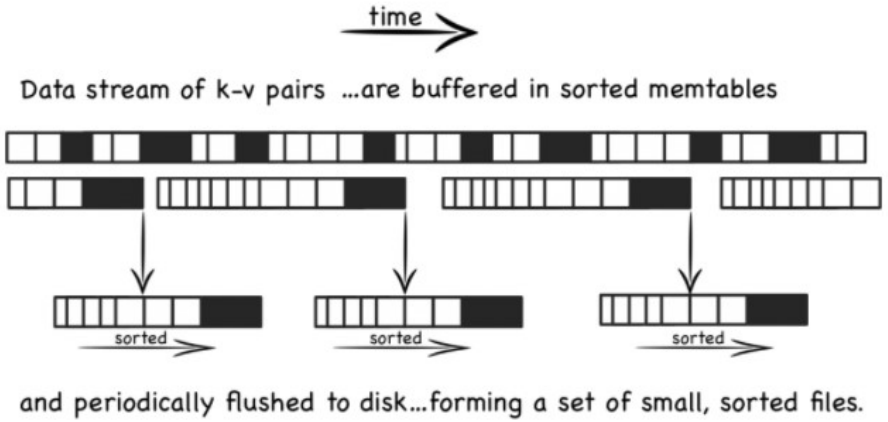
- 새로운 LSM Component 를 생성하며 Write
- Random I/O -> Sequential I/O 로 변환

Append only 로 저장된 이 데이터를 다시 효과적으로 읽을 수 있도록 재구성 (빠른 Read 성능)

- Back-ground Merge Process를 통해서 다시 Sequential Read 할 수 있도록 준비
- Random Read -> Sequential Write

LSM Tree 의 Merge 란 ?

- 데이터를 일단 작은 단위로 나누어 Sorting 하고
- 점점 더 큰 단위로 데이터를 생성하며 Merge



http://www.benstopford.com/2015/02/14/log-structured-merge-trees/

6.1 LSM-Tree 를 통해서 본 Clickhouse 의 특징

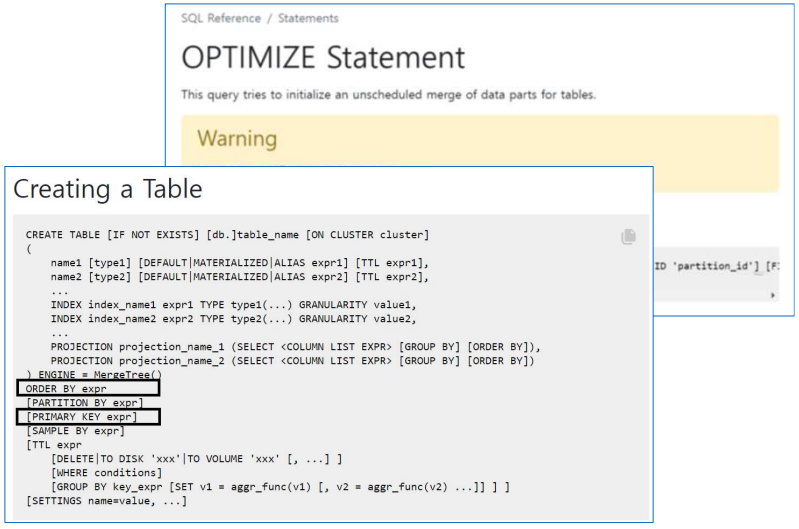
DEVIEW 2021

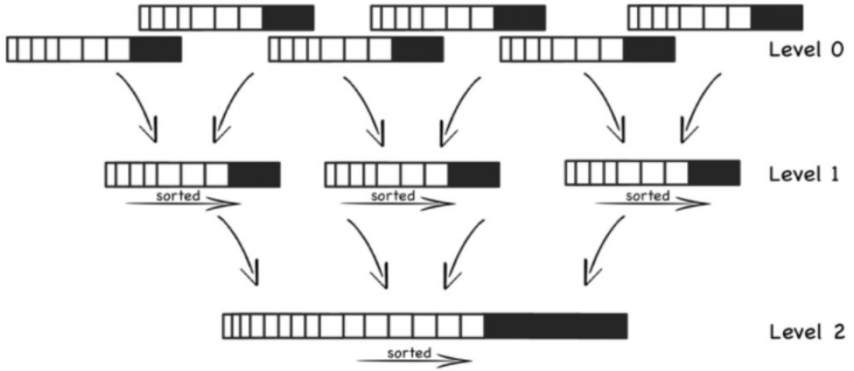
Merge Process 는 Write Performance 의 키

- 다시 읽을 때를 대비하여 Strictly Sort
- Primary Key 를 통해서 빠르게 Sequential Read

Merge Policy

- Cassandra 등 기타 LSM : Leveling Merge, Tiering Merge 와 같이 Merge Policy 를 선택 하는 것이 가능
- Clickhouse : Partition By 를 통해서 Merge Component 의 최대 크기를 조절 하는 것이 가능





http://www.benstopford.com/2015/02/14/log-structured-merge-trees/

6.1 LSM-Tree 를 통해서 본 Clickhouse 의 특징

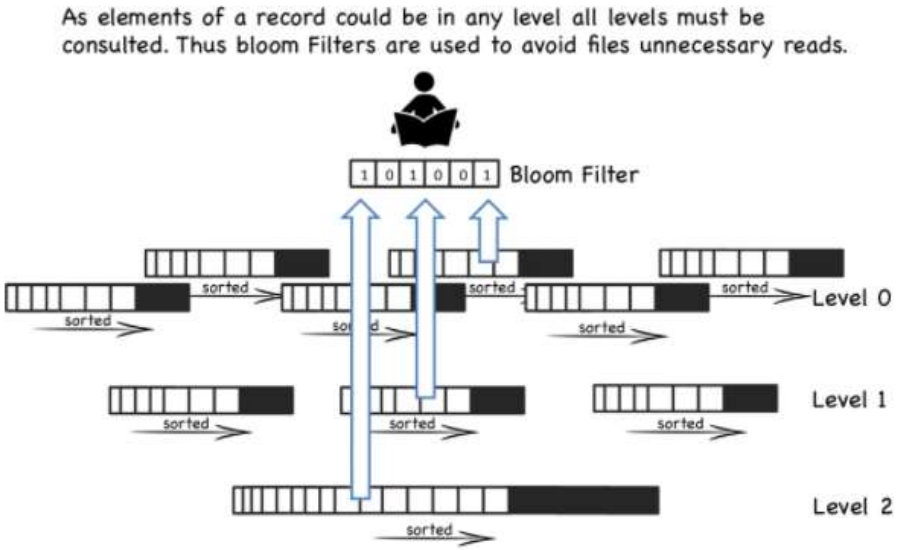
DEVIEW 2021

LSM Tree and Bloom Filter

- Bloom Filter 를 사용하여 수많은 LSM Component 중 Key data 의 위치를 찾아냄

Query 시에는 데이터 있는 Part 찾을 때에는 Bloom Filter

- 대상 데이터를 빠르게 찾아 낼 수 있음
- 단 이러한 특성으로 CH 의 경우 Write Visibility 가 떨어짐



http://www.benstopford.com/2015/02/14/log-structured-merge-trees/

6.1 LSM-Tree 를 통해서 본 Clickhouse 의 특징

DEVIEW 2021

Bloom Filter 의 기본적인 특정

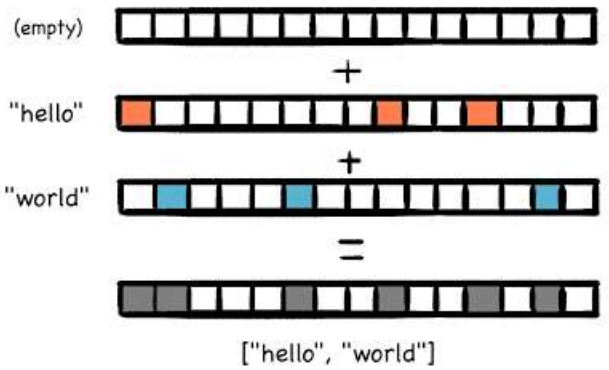
- 특정 원소 (Key) 가 집합 (LSM-Tree Component) 에 속하는지 아닌지를 검사하는 데에 사용
- 매우 빠르며 , Space-Efficiently
- False Positive (특정 key 가 집합에 속하지 않지만 있다고 할 가능성) 존재, 확률로서 Control 가능
- False Negative (있는데 없다고 할 가능성) 없음

Insert 시의 동작

- 데이터 마다 N 개의 Hash Function 을 사용하며 결과 값을 Bit Vector 에 계속해서 Or 연산
- N개의 함수이므로 N개이하의 비트가 세팅 된다 (Bit Vector Space 상 충돌 가능)

Search 시 동작

- 검색 대상 Key에 대해서 N 개의 Hash Function으로 연산하여 Bit Vector 를 구함
- 만약 Bit Vector 가 모두 켜져 있는 집합 (Component) 를 찾는다.
- False Positive 의 확률이 존재 (다음 Component 에서 Search)



예) 2 개의 데이터를 3개의 Hash Function 표현한 Bit Vector의 모습

https://steemit.com/kr-dev/@heejin/bloom-filter

6.2 Distributed Table Engine

DEVIEW 2021

자유로운 Table 조합을 가능하게 하는 Key Point

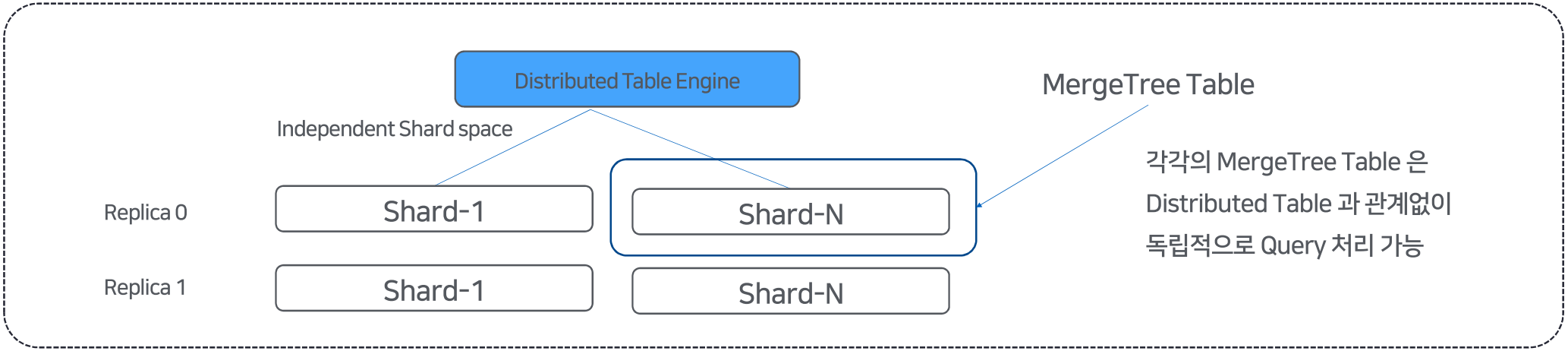
- MergeTree 를 Merge 한 관점을 제공

Sharding Key 를 통한 데이터 분배 및 H/A 를 담당하는 Table Engine

- 데이터 저장을 위한 테이블이 아닌 Sharding 및 H/A 를 구현한 구현체
- Distributed Table 을 통해 실제 데이터를 가지고 있는 MergeTree Engine table 로 Query 전달

MergeTree table 은 완벽히 독립적인 존재

- 이를 특정 Sharding key 와의 의존성 없이 자유롭게 Distributed 을 재구성할 수 있다.
- 이러한 이유로 Table 생성을 2번 하는 것

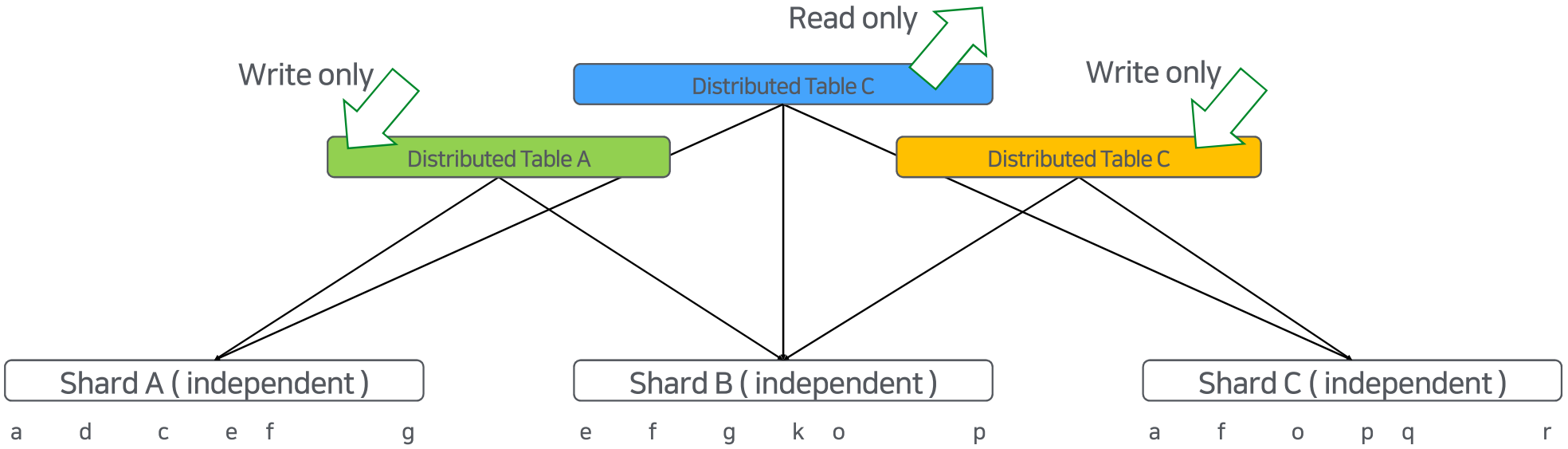


6.2 Distributed Table Engine

Query 가능한 독립적 LSM Component 의 특성이 MergeTree 까지 반영되어 있다.

Table Schema 만 같다면 다양하게 Shard 를 재조합 하여 사용할 수 있다.

이러한 특성을 이용하여 Global IDC 의 다양한 Topology 를 구성하는데 사용할 수 있다.

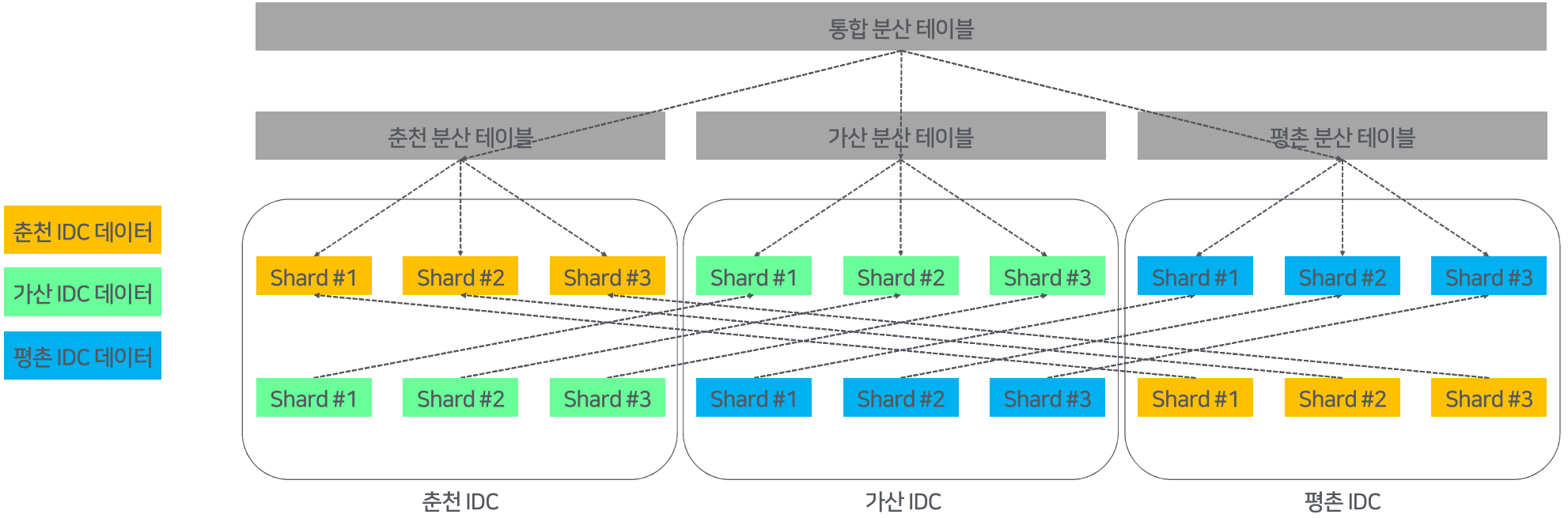


6.3 기타 다양한 응용

DEVIEW 2021

Cross IDC Replication

- IDC 간의 데이터를 복제
- 하나의 IDC 가 장애 자연재해 등과 같은 장애를 일으켜도 서비스 가능

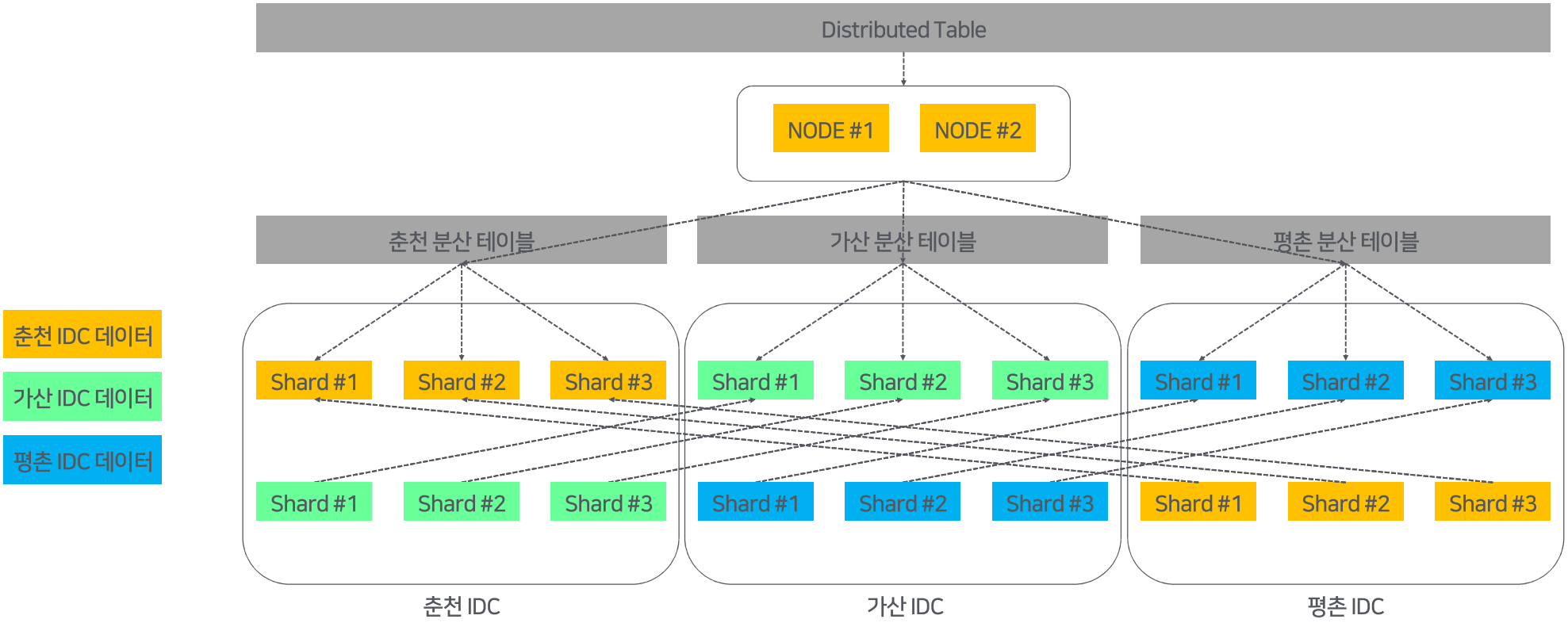


6.3 기타 다양한 응용

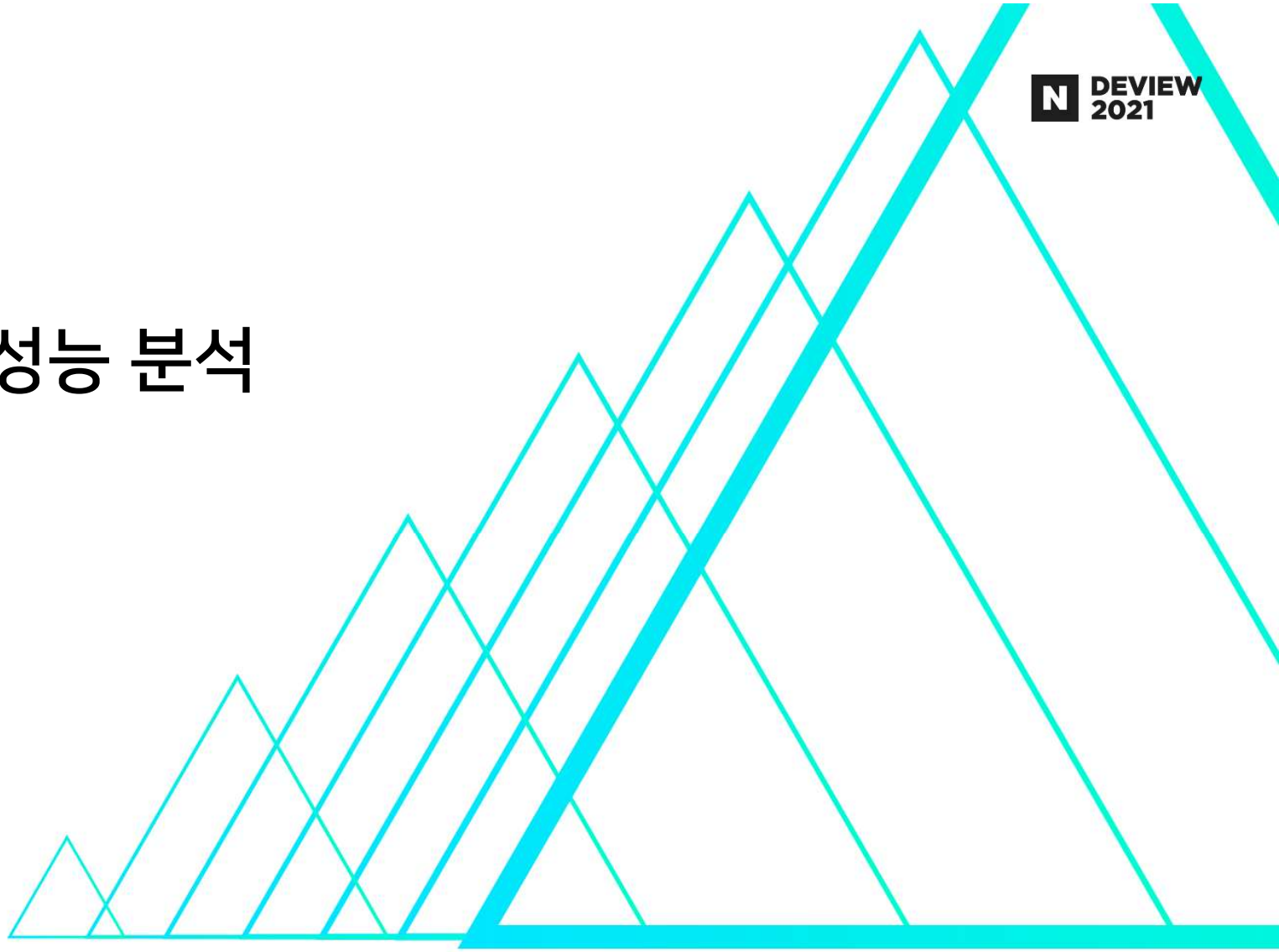
DEVIEW 2021

Read Only Cluster

- 데이터를 가지고 있지 않은 Node 에서 Distributed Table 이용
- 최종 Aggregation 이 많이 발생하는 경우



7. 성능 분석



7.1 성능 측정 (통합 테이블)

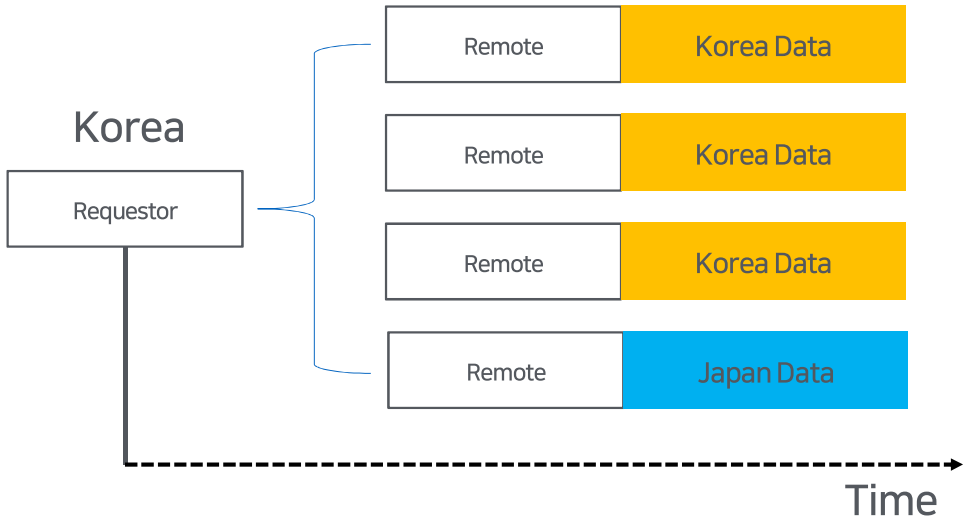
DEVIEW 2021

성능 변수들

- IDC 별 데이터 볼륨,
- IDC 간의 망 속도,
- Server 의 성능

Requestor Node/ Remote 노드 구분

- Requestor Node 는 한국 IDC 내 존재
- 한국과 일본데이터를 모두 대상으로 함.



측정된 성능 특성

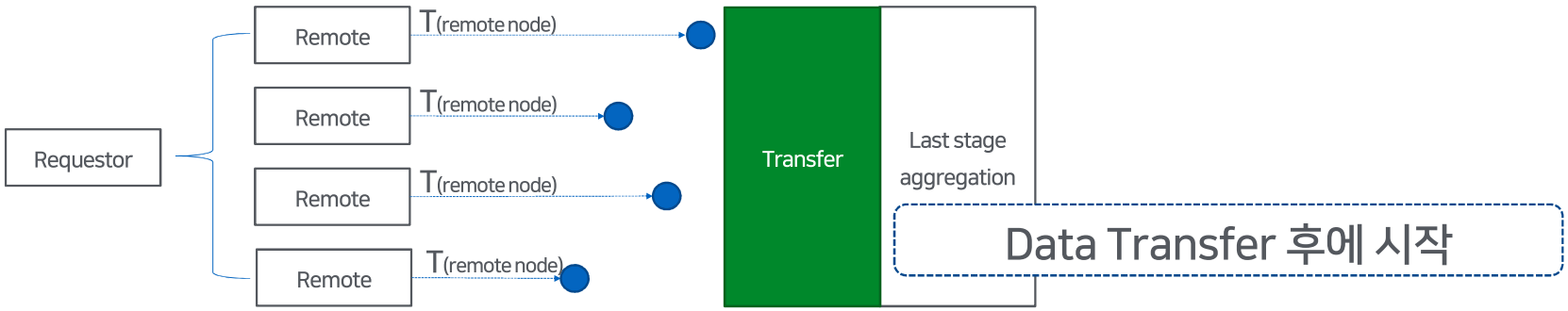
- 한국, 일본 데이터를 모두 통합하여 Query 하는 경우 성능 변화는 거의 없다
- 매우 Light 한 Query를 하는 경우 30ms -> 600ms 의 비교적 큰 변화 관측
- 왜 이런 일이 일어나는 걸까 ?

7.1 성능 측정 (통합 테이블)

DEVIEW 2021

Clilckhouse의 MergeTree 특성상 MergeTree 에 대한 Query 실행은 병렬화

- MergeTree engine 의 데이터 처리 과정



수식화

$$\text{전체 Query 실행 시간} \quad T_{total} = T_{max(remote\ node)} + T_{transfer} + T_{(Last\ stage\ aggregation)}$$

병렬처리와 IDC간의 Data Skew로 적은 Volume의 IDC 데이터는 성능에 큰 영향을 미치지 않음

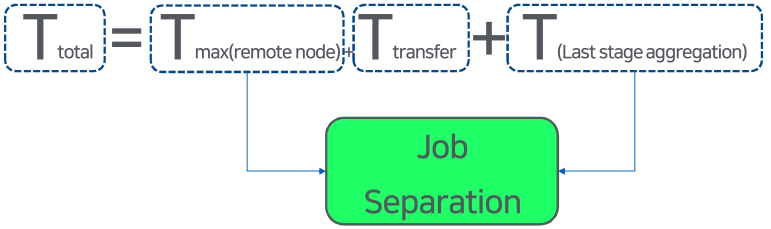
7.2 성능의 주요 요소

DEVIEW 2021

Remote/Requestor Node Separation

- Requestor Node : Query 를 실행하는 Node
- Remote Node : 데이터를 실제 가지고 있는 원격 Node
- 이 두개를 분리하는 Configuration 을 적용 하여 실험

전체 Query 실행 시간



성능 향상은 미미

- Remote Node 가 Requestor Node 가 되는 것이 더 좋겠다는 판단.
- Query 실행에 필요한 메모리만 충분하다면 이 둘을 분리하지 않는 것도 방법
- 병렬처리 되는 Query 중 1개는 Local 데이터 전송을 통해 처리하는 셈.
- 따라서 Memory 만 충분하다면 특별히 느릴 이유가 없음

LCS Query 성능 비교

- **Query 1: 합산과 Uniq**
  - select hour, toInt64(sum (pv / length(users))) , uniq(arrayJoin(users)) from dlcs_stms_array_basic_20210629 where (stname like 'NHN/한국/네이버/모바일_웹/%') AND (age != '-') GROUP BY hour ORDER
- **Query 2: 합산**
  - select hour, toInt64(sum (pv / length(users))) from dlcs_stms_array_basic_20210629 where (stname like 'NHN/한국/네이버/모바 일_웹/%') AND (age != '-') GROUP BY hour ORDER BY hour ASC
- **Query 3: AMT 합산과 Uniq**
  - SELECT hour, toUInt64(sumMerge(pv)), uniqMerge(uv) FROM damtlcs_stms_array_basic_20210629 WHERE (stname like 'NHN/한국/네이버/모바일_웹/%') AND (age != '-') GROUP BY hour ORDER BY hou

	LCS Q1		LCS Q2		LCS Q3	
Try #	Read and Write Node	ReadOnly	Read and Write Node	ReadOnly	Read and Write Node	ReadOnly
1	3.031	2.543	1.873	2.144	3.534	3.033
2	2.716	2.551	1.984	2.39	2.973	3.402
3	2.927	2.834	1.624	2.05	3.14	3.109
4	2.994	2.48	2.104	2.24	3.382	3.027
5	2.657	2.932	2.164	2.457	3.461	3.119
6	2.434	2.612	2.114	1.91	3.305	3.055
7	2.991	1.935	1.62	2.109	3.337	2.977
8	2.074	2.559	1.985	2.241	3.777	3.19
9	2.674	2.854	2.194	2.428	3.243	3.023
10	2.517	2.023	2.405	1.617	3.819	3.473
11	2.669	2.844	2.11	2.397	3.55	3.427
12	2.964	2.583	1.769	1.949	3.622	3
13	2.964	1.981	1.942	2.437	3.098	3.144
14	2.747	2.605	1.946	2.165	3.288	3.378
15	2.432	2.51	1.984	2.127	3.28	3.098
16	2.623	2.547	1.607	1.61	3.306	2.759
17	2.713	2.493	2.061	2.009	3.087	3.083
18	2.121	2.037	2.241	1.954	3.637	3.082
19	2.709	2.555	2.392	1.584	3.174	2.915
20	3.04	2.506	2.423	1.707	3.591	2.805
Average	2.69985	2.4992	2.0271	2.07625	3.3802	3.10495

7.2 성능 요소 요약

Real-time OLAP을 처리 중인 현재의 시스템

- 해외/국내 네이버 전체 사용자를 대상으로 한 다양한 로그
- Flink 를 통한 1st Aggregation 데이터를 Window Triggered Write
- 일간 14억 레코드 (1750 records/sec) 를 Write 하며 Read

IDC 별 Data Skewness 를 고려한 전략적 Requestor Node 선택

- Requestor Node 의 선택이 중요 (Data 가 많은 IDC 에서 Query 를 실행 해라)
- Aggregate Pushdown의 특성을 고려하여 Requestor Node 를 선택하는 것이 현명함.
- IDC 간의 Data Skewness 를 적극적으로 이용하기

DEVIEW 2021

8. Wrap-up

8.1 영상을 마치며...

DEVIEW 2021

Data Protection Law 의 중요성

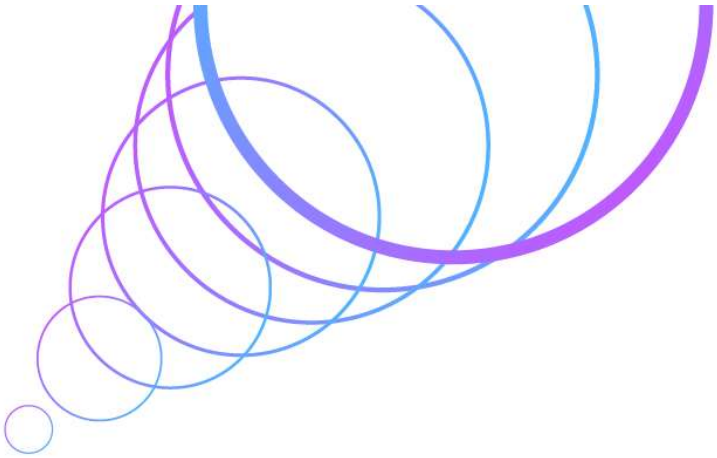
- 회사와 개인에게 심각한 Penalty 를 초래 할 수도
- Global Business Launching 시에는 더욱 중요

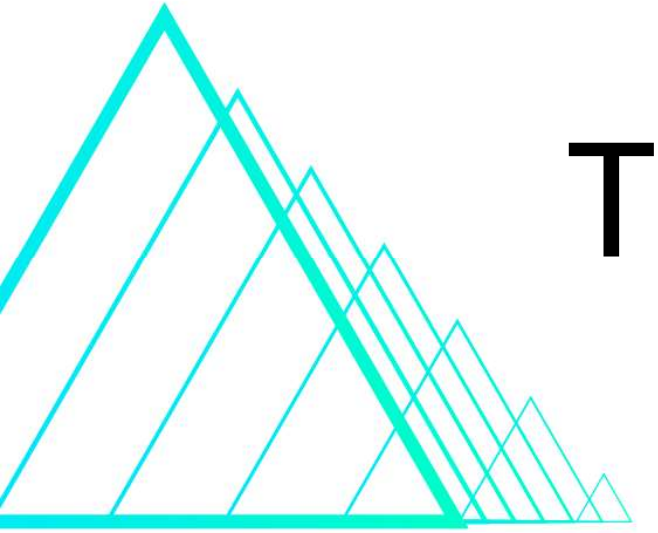
https://www.good-id.org/en/articles/new-age-data-privacy-3-core-concepts-privacy-laws-around-world/

토종 국내 기업도 이제는 글로벌로 많은 확장을 시도 하고 있습니다.
하지만, 국가 마다 많은 법규와 허들로 인해 주춤하는 경우도 많을 것이라 생각됩니다.

많은 허들 중에 데이터 주권주의, 개인정보보호법 등으로 인해 Data Localization(현지화) 문제를 저희는 Clickhouse의 특성을 이용해서 풀었던 내용을 공유 드렸습니다.

이 영상을 통해 유사한 고민을 하고 계시는 기업이나 개발자분들에게 도움이 되셨으면 좋겠습니다.





Thank You