DEVIEW 2020

COVID19시대 빠른 사용자 행동 패턴 파악

김영진 이모원 박종민 NAVER INSIGHT CENTER / Log data

CONTENTS

DEVIEW 2020

1. COVID-19가 가져온 변화

CHANGE

1.1 COVID19로 인한 변화 (1/2)

DEVIEW 2020

사용자의 네이버 사용 패턴 변화

- 매일 아침 8시 59분, **마스크 대란**
- 실시간 안전안내문자, **확진자 현황 및 동선 확인**
- 국가 지원금 **대상자 확인**
- ...

1.1 COVID19로 인한 변화 (2/2)

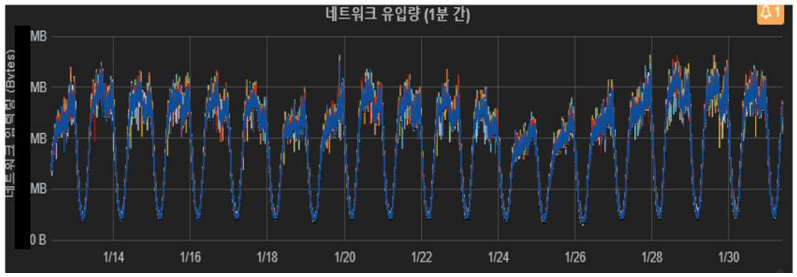
DEVIEW 2020

네이버의 변화

- 언택트 시대, 비대면서비스, **테이블주문, QR결제 ...**
  - 서비스 retention

- 적극적인 마케팅 , **효과적인 마케팅에 집중**
  - TV 광고 직후 conversion

2020 년 1월



2020년 3월

covid19 전과 후의 변화

[Batch집계에서 볼 수 없는 인사이트 등장]

2. 필요한 실시간지표시스템의 방향

NEED

2.1 단순집계를 넘어 OLAP 형태의 실시간 지표

DEVIEW 2020

실시간 지표를 다양한 dimension/metric 형태로 제공

- OLAP 기반의 Backend 아키텍쳐 구성
- 단순 지표가 아닌 다양한 세그먼트로 세분화
- 쿼리형태 보다 의미 있는 조합의 지표를 보다 쉬운 인터페이스로 제공

굉장히 다양한 DIMENSION / METRIC의 조합으로 인한 빠른 처리와
효과적인 저장 그리고 지연없이 서빙 할 수 있는 기술력 & 아이디어 필요

[네이버 실시간 지표 시스템 엿보기]

2.2 다양한 집계 단위 구성

DEVIEW 2020

서비스 단위의 집계를 넘어 개별 페이지 단위의 집계

- 네이버전체 / 서비스 / 서비스파트 / 서비스세부파트 / 페이지URL 의 집계 단위
- 개별 페이지 개선 시 빠르고 정확한 사용자 피드백 확인
- 페이지 URL은 지금 이 순간에도 늘어나고 있는 항목

괴장히 많은 Cardinality를 효과적으로 대응 할 수 있는 기술력 & 아이디어 필요

[네이버 실시간 지표 시스템 엿보기]

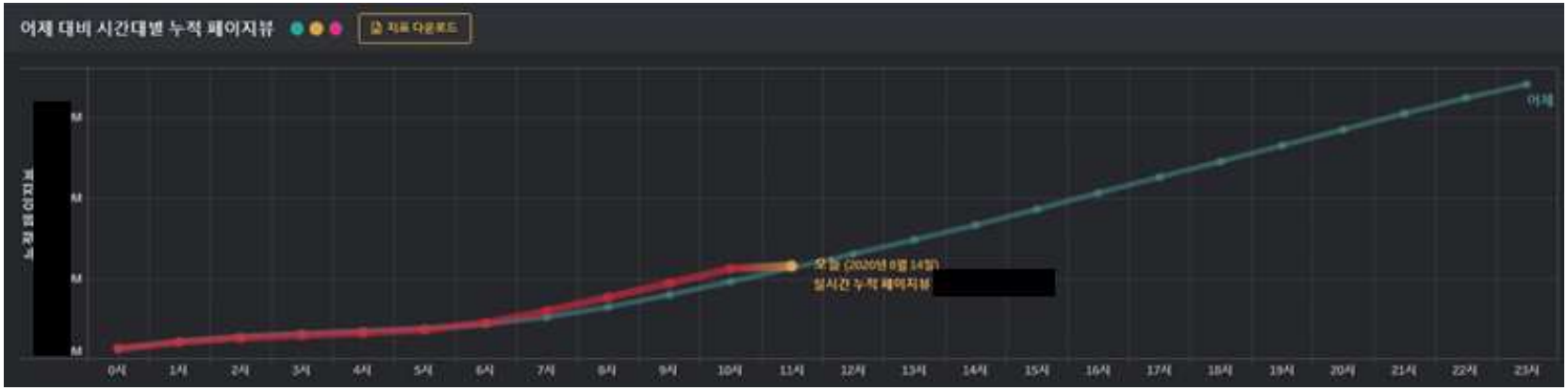
2.3 단순지표 제공이 아닌 추이/추세 형태

DEVIEW 2020

실시간의 경우 지표시스템이 정한 짧은 윈도우에 따른 값이라는 특징

- 단일 값 자체에 대한 의미 부여보다는 긴 시간 동안의 그래프 모습에 집중
- 전일, 전주, 전월 등 동일 시점과 추이/추세를 비교했을 경우 직관적인 의미 부여가능

과거 긴 기간의 dimension/metric 조합의 값을 실시간 지표와 함께 서빙이 가능한 기술력 & 아이디어 필요



[네이버 실시간 지표 시스템 엿보기]

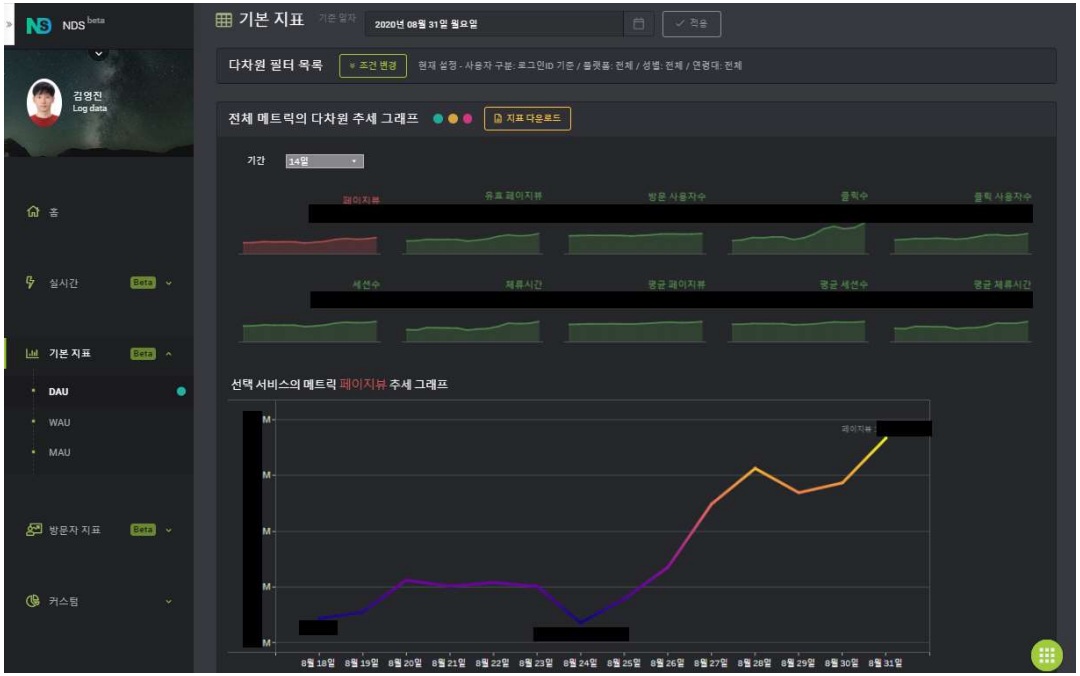
2.4 지표의 의미/속성을 잘 표현하는 화면개발

DEVIEW 2020

지표의 의미와 속성에 맞는 표현은 필수

- 퀄리티 있는 템플릿을 기본 베이스로 UX/UI 구성
- Backend의 cube 성능 극대화 & 부족한 표현력을 메꿔 줄 수 있는 BI 도구 활용

좋은 아이디어로 이미 만들어져 있는 템플릿을
BI도구와 함께 활용할 수 있는
기술력 & 아이디어



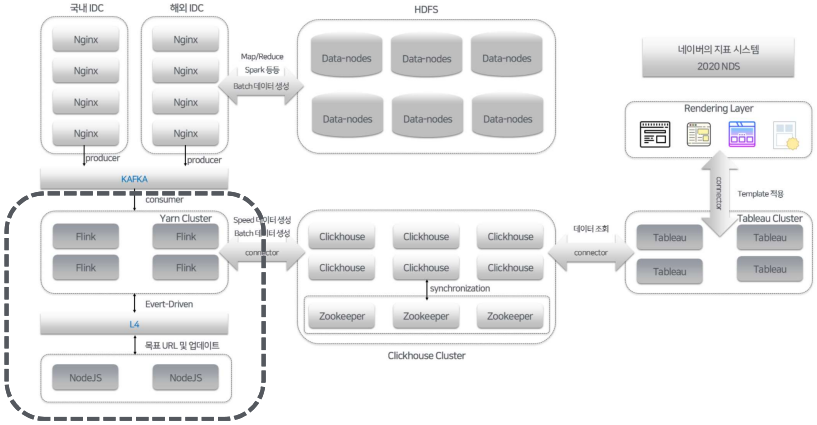
[네이버 실시간 지표 시스템 엿보기]


지금까지 말씀드린
부분들을 어떻게
해결했는지 그 내용을
자세히 설명 드리겠습니다.

DEVIEW 2020

3. 실시간 데이터 처리파트

SPEED



3.1 Flink를 이용한 실시간 데이터 처리

DEVIEW 2020

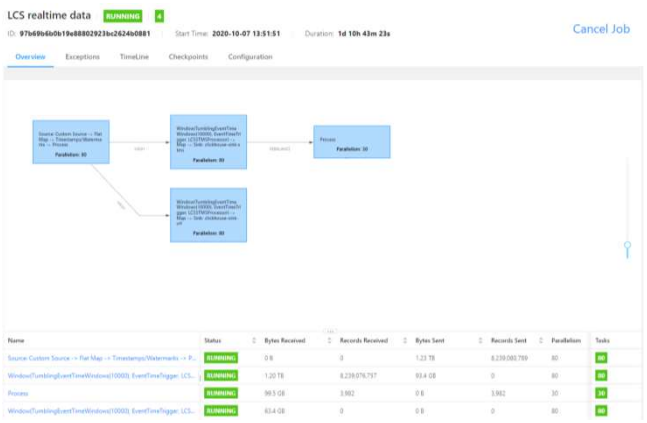
Flink 를 이용한 실시간 데이터 처리 시스템

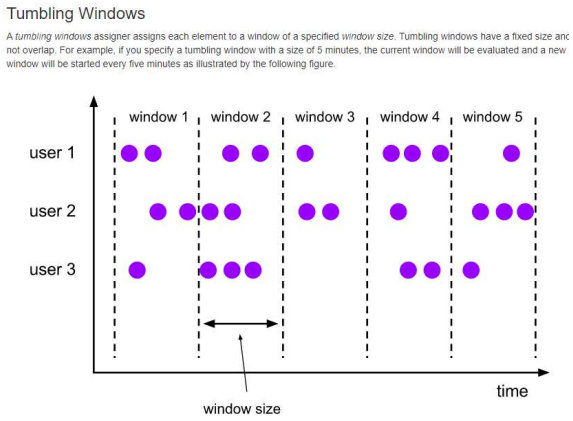
- Micro Batch가 아닌 True streaming/Low latency가 주는 빠른 성능
- 3년간의 개발/운영 경험, 다양한 활용 분야 적용
- 버전 업이 가져다 주는 Dramatic 한 성능 개선 하나의 이점
- 최근의 업데이트에서 주요 사항

Flow Control 의 개선

Dashboard 개선

Back Pressure monitoring 도 개선



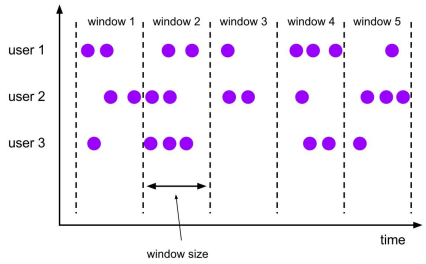


3.2 Flink's Window and Trigger (1/2)

DEVIEW 2020

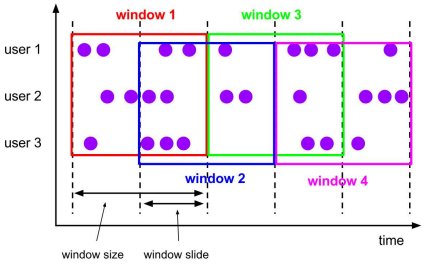
Tumbling Window (Fixed time windows)

- 특정 시간 Interval 마다 Windows Trigger (Fire)
- Overlap 없음



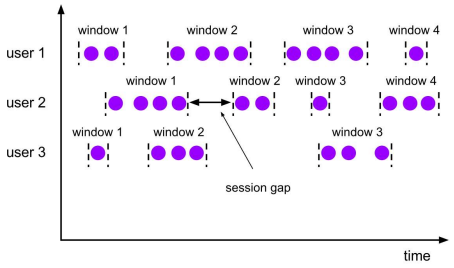
Sliding Window

- 특정 시간 Interval 마다 Window Triggered
- Overlap 을 설정하여 데이터를 좀 더 넓게 볼 수 있음



Session Window

- 사용자 Session 이 활동을 멈추었을 경우 Triggered
- 세션단위의 분석을 하는 경우 매우 효과적으로 사용가능



3.2 Flink's Window and Trigger (2/2)

DEVIEW 2020

Custom Trigger

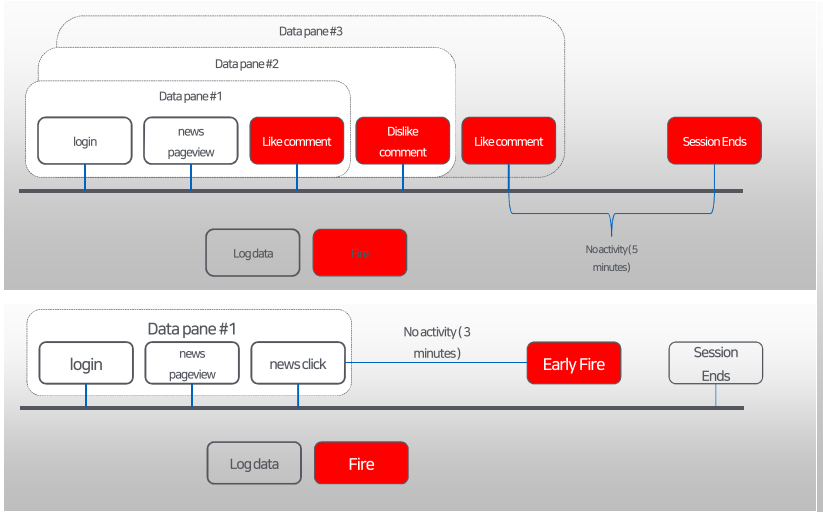
- Flink의 가장 강력한 도구 중에 하나
- 원하는 시간, 이벤트로 Trigger된 Window 생성 후, Data 분석

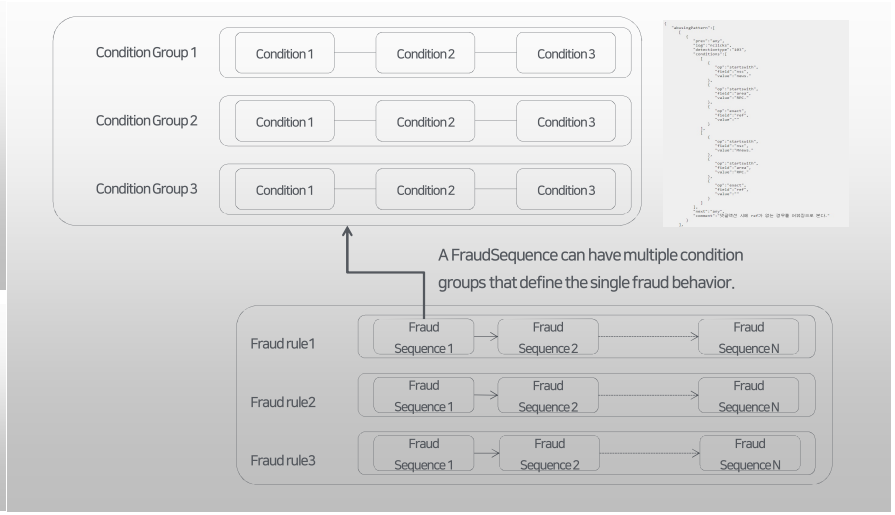
Event 를 기반 Trigger

Early Triggering

Sequence based Trigger

- 다양한 Windowing 가능





3.3 Late data 처리에 대한 고민

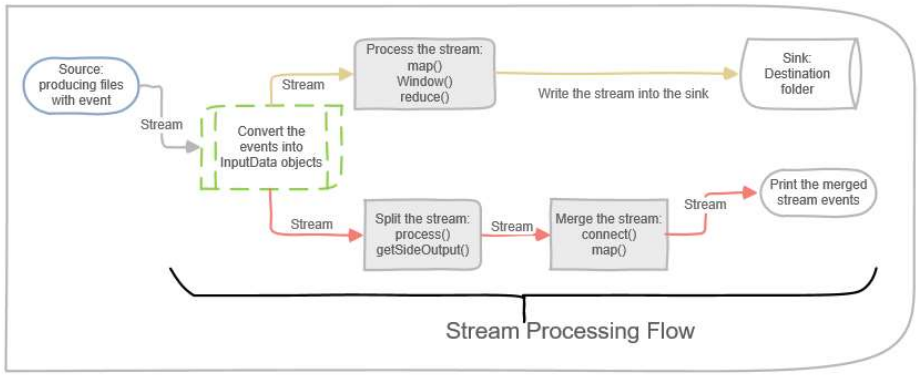
DEVIEW 2020

Side output 처리 혹은 allowedLateness

- 늦게 도착한 데이터에 대한 별도의 처리 (sideoutput)
- LateFiring 을 통한 결과 업데이트(allowedLateness) – 중복 데이터 구분 구조 필요

late data 처리를 통해서 소량의 지표 오차도 검출

- 특정 Threshold 이상일 경우, 지표 복구를 수행하도록 자동화



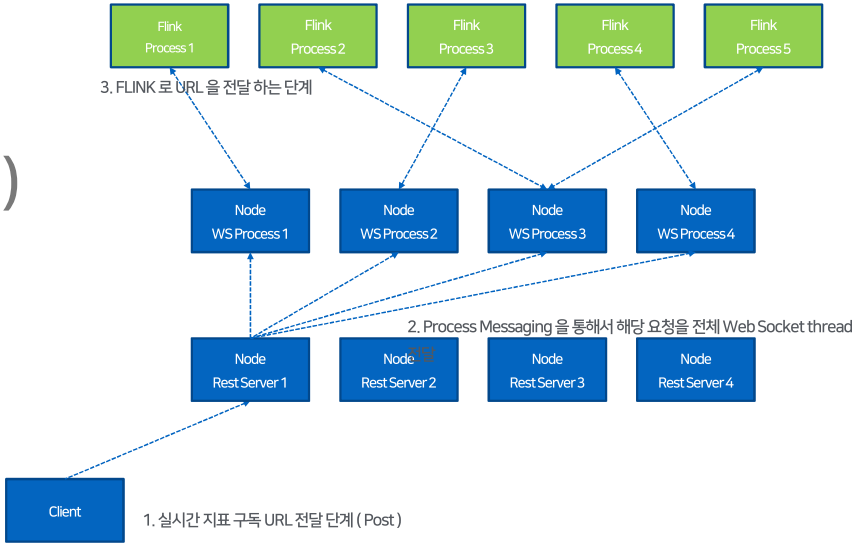
3.4 선택적 집계를 위한 API 구성 방법

DEVIEW 2020

외부 서버와 통신하기 위한 API 구조

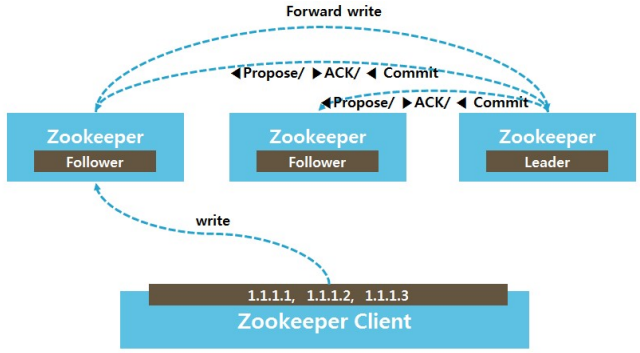
- Target URL 정보를 동시에 빠르게 전송 (Realtime)
- IP 가 바뀌는 Flink 가 Client (Yarn)
- Polling은 비효율적

적절한 Polling interval 값 찾기의 어려움
동기화 시간차



Node, IPC, WebSocket 로 개발

- Subscribe/Publish 구조 (Event-Driven)
- Zookeeper 이용도 하나의 방법 (Ensemble)

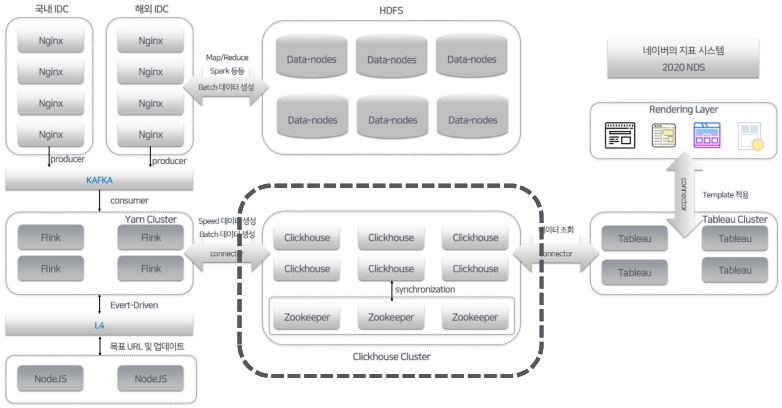


이미지 출처: d2.naver.com/helloworld/583580

DEVIEW 2020

4. 실시간 데이터 조회파트

OPTIMIZE



4.1 Clickhouse와 실시간 OLAP

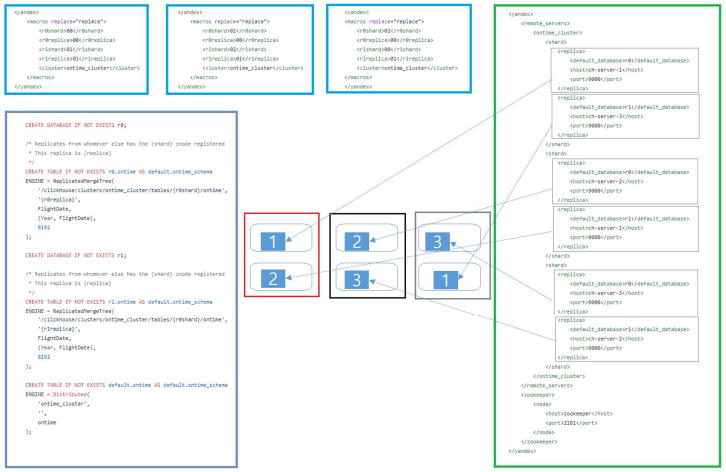
DEVIEW 2020

Clickhouse의 특징

- 매우 빠른 OLAP DBMS
- 적은 수의 Node 만으로도 훌륭한 성능
- 간단하고 직관적인 Cluster setup
- H/W 추가/업그레이드 성능 개선 폭 큼(H/W efficient)
- 단일 DBMS 형태

Hadoop등과의 연동은 외부 Library 이용

3rd party tools/lib 는 점차 많아지는 중

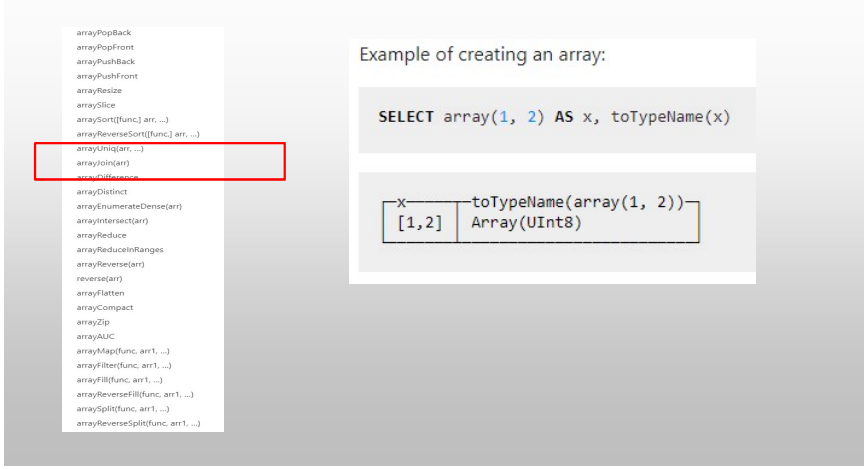


4.2 Clickhouse의 Array&ArrayJoin (1/4)

DEVIEW 2020

Clickhouse Array

- Native DataType
- 다양한 함수로 활용 가능



Array Join

- 매우 특이하지만 활용 가능성이 높은 함수
- Array Element를 Row로 Converting (Col->Row)

arrayJoin function ¶

This is a very unusual function.

Normal functions don't change a set of rows, but just change the values in each row (map).
Aggregate functions compress a set of rows (fold or reduce).
The 'arrayJoin' function takes each row and generates a set of rows (unfold).

Example:

```
SELECT arrayJoin([1, 2, 3] AS src) AS dst, 'Hello', src
```

```
┌─dst─┬─\'Hello\'─┬─src─────┐
│   1 │ Hello     │ [1,2,3] │
│   2 │ Hello     │ [1,2,3] │
│   3 │ Hello     │ [1,2,3] │
└─────┴───────────┴─────────┘
```

4.2 Clickhouse의 Array&ArrayJoin (2/4)

DEVIEW 2020

Clickhouse Array의 활용

- Clickhouse Array 와 ArrayJoin을 활용하여 다음과 같이 ROW를 축소
- PV 값은 sum(), UV 의 경우 Array 형태로 Merge
- 필요시 ArrayJoin() 를 이용하여 Row 형태로 원복

2020-09-01	12:03:10	12:03:20	네이버 블로그	30-40	F	Android	Browser	1	사용자A
2020-09-01	12:03:10	12:03:20	네이버 블로그	30-40	F	Android	Browser	3	사용자B
2020-09-01	12:03:10	12:03:20	네이버 블로그	30-40	F	Android	Browser	4	사용자C
2020-09-01	12:03:10	12:03:20	네이버 블로그	30-40	F	Android	Browser	5	사용자D

Array 로 표현

2020-09-01	12:03:10	12:03:20	네이버 블로그	30-40	F	Android	Browser	13	[사용자A,B,C,D]

ArrayJoin()

2020-09-01	12:03:10	12:03:20	네이버 블로그	30-40	F	Android	Browser	13	사용자A
2020-09-01	12:03:10	12:03:20	네이버 블로그	30-40	F	Android	Browser	13	사용자B
2020-09-01	12:03:10	12:03:20	네이버 블로그	30-40	F	Android	Browser	13	사용자C
2020-09-01	12:03:10	12:03:20	네이버 블로그	30-40	F	Android	Browser	13	사용자D

4.2 Clickhouse의 Array&ArrayJoin (3/4)

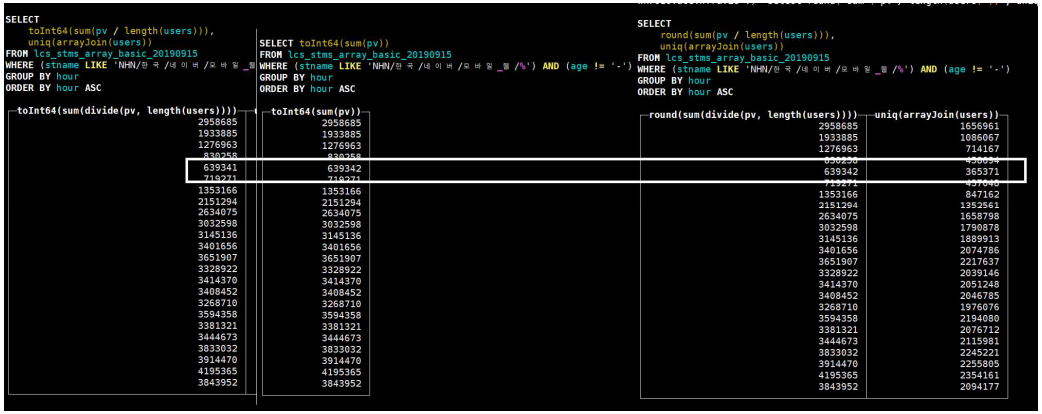
DEVIEW 2020

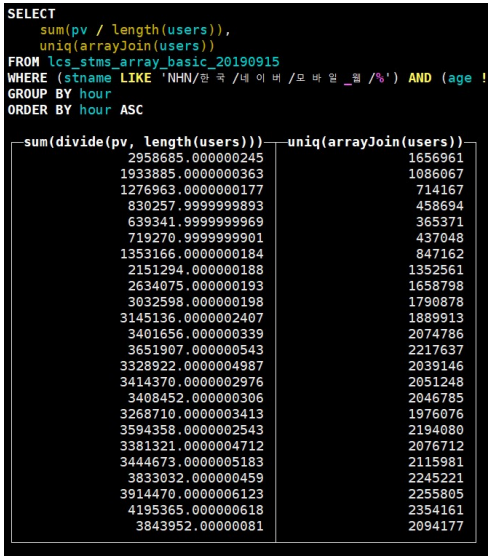
Clickhouse ArrayJoin의 활용

- User length만큼 PV 를 나누어 Query 사용가능

```
select hour, toUInt64(sum(pv / length(users))) AS pv , uniq(arrayJoin(users)) FROM dlcs_stms_array_basic_20200919
WHERE stname='통합검색' GROUP BY hour ORDER BY hour asc
```

- 과정에서 나눗셈으로 인한 소수점이 발생
- Round() 함수를 이용하여 개선이 가능





```
SELECT
    sum(pv / length(users)),
    uniq(arrayJoin(users))
FROM lcs_stms_array_basic_20190915
WHERE (stname LIKE 'NHN/한 국 /네 이 버 /모 바 일 _웹 /%') AND (age !
GROUP BY hour
ORDER BY hour ASC
```

sum(divide(pv, length(users)))	uniq(arrayJoin(users))
2958685.000000245	1656961
1933885.0000000363	1086067
1276963.0000000177	714167
830257.9999999893	458694
639341.9999999969	365371
719270.9999999901	437048
1353166.0000000184	847162
2151294.000000188	1352561
2634075.000000193	1658798
3032598.000000198	1790878
3145136.0000002407	1889913
3401656.000000339	2074786
3651907.000000543	2217637
3328922.0000004987	2039146
3414370.0000002976	2051248
3408452.000000306	2046785
3268710.0000003413	1976076
3594358.0000002543	2194080
3381321.0000004712	2076712
3444673.0000005183	2115981
3833032.000000459	2245221
3914470.0000006123	2255805
4195365.000000618	2354161
3843952.00000081	2094177

4.2 Clickhouse의 Array&ArrayJoin (4/4)

DEVIEW 2020

Clickhouse Array의 활용

- Row수를 줄여 Disk Access 성능 개선 가능
- 필요시 ArrayJoin() 를 이용하여 Row 원복
- 계산방법에 따라서 Row를 바꿔가며 계산

Clickhouse Array 의 활용

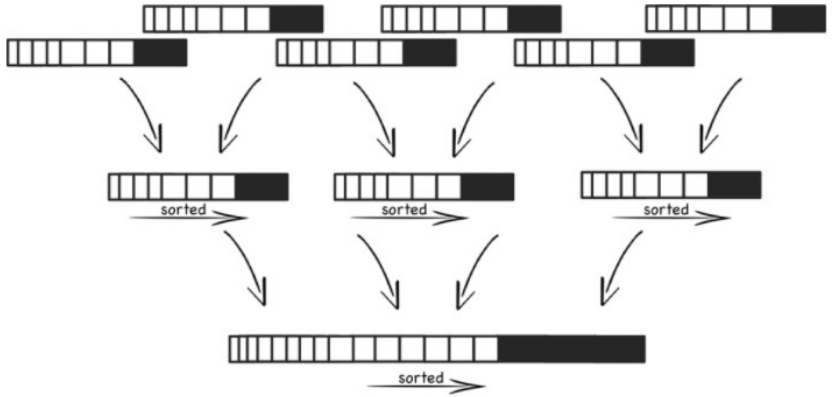
- 일단 35억 Records -> 5억 Records
- 10초 단위의 실시간 OLAP 성능 확보 (6시간이상도 훌륭)
- 그러나 일간 단위의 지표 제공, 전일 비교를 위해선 추가 개선이 필요

4.3 AggregatingMergeTree (1/3)

DEVIEW 2020

MergeTree Engine 동작에 대한 기본 설명

- C.H의 INSERT는 새로운 Part 를 생성하며 저장
- (Insert Throughput 향상, I/O 측면에서 Write Buffer 역할)
- Background Job이 기존의 Part와 Sorting 하여 하나의 Part로 Merge
- AMT는 같은 키를 가진 데이터를 Merge 하며 별도의 연산을 가능하게 함



Compaction continues creating fewer, larger and larger files

f(S1 UNION ALL S2) = f(f(S1) UNION ALL f(S2)).

4.3 AggregatingMergeTree (2/3)

DEVIEW 2020

AggregatingFunction

- Merge시 동일한 Primary Key를 가진 Row 병합
- SimpleAggregateFunction 을 사용하여 min, max, sum 등의 최종 값 획득
- AggregateFunction 을 사용하여 intermediate binary 값 저장

f(S1 UNION ALL S2) = f(f(S1) UNION ALL f(S2))

이 조건이 만족 된다면

- SimpleAggregateFunction

그렇지 않다면

- AggregateFunction

Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2],
    ...
) ENGINE = AggregatingMergeTree()
[PARTITION BY expr]
[ORDER BY expr]
[SAMPLE BY expr]
[TTL expr]
[SETTINGS name=value, ...]
```

- Allow using groupArrayArray and groupUniqArrayArray as SimpleAggregateFunction

4.3 AggregatingMergeTree (3/3)

DEVIEW 2020

예제

```
CREATE TABLE URL_PAGEVIEW
(
    `MetricDate` date,
    `URL` String,
    `PV` SimpleAggregateFunction(sum, UInt64)
)
ENGINE = AggregatingMergeTree()
ORDER BY (MetricDate, URL)
```

1. 테이블 생성

```
xkr010.ace.nfra.io :) insert into URL_PAGEVIEW values('2020-10-01', 'https://m.naver.com', 4 )
xkr010.ace.nfra.io :) insert into URL_PAGEVIEW values('2020-10-02', 'https://m.naver.com', 3 )
xkr010.ace.nfra.io :) insert into URL_PAGEVIEW values('2020-10-01', 'https://m.naver.com', 4 )
xkr010.ace.nfra.io :) insert into URL_PAGEVIEW values('2020-10-01', 'https://m.naver.com', 2 )
xkr010.ace.nfra.io :) insert into URL_PAGEVIEW values('2020-10-01', 'https://www.naver.com', 1 )
xkr010.ace.nfra.io :) insert into URL_PAGEVIEW values('2020-10-02', 'https://www.naver.com', 4 )
xkr010.ace.nfra.io :) insert into URL_PAGEVIEW values('2020-10-02', 'https://www.naver.com', 5 )
xkr010.ace.nfra.io :) insert into URL_PAGEVIEW values('2020-10-02', 'https://www.naver.com', 4 )
```

2. 데이터 준비

```
SELECT *
FROM URL_PAGEVIEW

┌─MetricDate─┬─URL─────────────────┬─PV─┐
│ 2020-10-01 │ https://m.naver.com │ 10 │
│ 2020-10-01 │ https://www.naver.com │ 1 │
│ 2020-10-02 │ https://m.naver.com │ 3 │
│ 2020-10-02 │ https://www.naver.com │ 4 │
└────────────┴─────────────────────┴────┘

┌─MetricDate─┬─URL─────────────────┬─PV─┐
│ 2020-10-02 │ https://www.naver.com │ 5 │
└────────────┴─────────────────────┴────┘

┌─MetricDate─┬─URL─────────────────┬─PV─┐
│ 2020-10-02 │ https://www.naver.com │ 4 │
└────────────┴─────────────────────┴────┘
```

3. INSERT 이후

```
OPTIMIZE TABLE URL_PAGEVIEW FINAL

Ok.
```

4. Merge

```
SELECT *
FROM URL_PAGEVIEW

┌─MetricDate─┬─URL─────────────────┬─PV─┐
│ 2020-10-01 │ https://m.naver.com │ 10 │
│ 2020-10-01 │ https://www.naver.com │ 1 │
│ 2020-10-02 │ https://m.naver.com │ 3 │
│ 2020-10-02 │ https://www.naver.com │ 13 │
└────────────┴─────────────────────┴────┘
```

5. 최종 상태

4.4 AMT + Materialized View (1/3)

DEVIEW 2020

Clickhouse의 Materialized View

- Query의 결과 값으로 별도의 Table 생성하는 Materialized View
- Clickhouse 에서는 다양한 MergeTree 와 혼합하여 독특하게 동작
- Source Table 의 Insert 에 Trigger로 동작 하는 개념

```
CREATE MATERIALIZED VIEW r0.amtlcs_stms_array_basic_20201009
(
    `date` Date,
    `hour` UInt32,
    `stname` String,
    `origin` String,
    `age` String,
    `gender` String,
    `os` String,
    `browser` String,
    `pv` AggregateFunction(sum, Float64),
    `uv` AggregateFunction(uniq, String)
)
ENGINE = ReplicatedAggregatingMergeTree('/clickhouse/clusters/{cluster}/tables/amtlcs_stms_array_basic_20201009_{r0shard}', '{r0replica}')
PARTITION BY toYYYYMM(date)
ORDER BY (stname, origin, hour, age, gender, os, browser, date)
SETTINGS index_granularity = 8192 AS
SELECT
    date,
    hour,
    stname,
    origin,
    age,
    gender,
    os,
    browser,
    sumState(pv / length(users)) AS pv,
    uniqState(arrayJoin(users)) AS uv
FROM r0.lcs_stms_array_basic_20201009
GROUP BY
    date,
    hour,
    stname,
    origin,
    age,
    gender,
    os,
    browser
```

Aggregating 시에 사용될 연산 종류

Merge 에 사용될 Primary Key

Insert Trigger 시 source table 에서 수행할 SELECT

4.4 AMT + Materialized View (2/3)

DEVIEW 2020

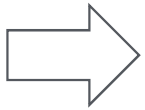
Clickhouse의 Materialized View

- Source Table 과는 별도
- 시간,일 단위 데이터를 위해 Merge
- 큰 폭으로 Records 수를 감소 가능

- 5억2천만개의 Records -> 2500 만개 Records 를 단위로 Merge
- 원본 35억 -> 2천만

```
SELECT count(1)
FROM dlcs_stms_array_basic_20201008

┌─count(1)─┐
│ 525998880 │
└──────────┘
```



```
SELECT count(1)
FROM damtlcs_stms_array_basic_20201008

┌─count(1)─┐
│ 25467452 │
└──────────┘
```

4.4 AMT + Materialized View (3/3)

DEVIEW 2020

Source 테이블에서 다양한 단위로 재집계 가능

- 초 단위의 집계 단위는 많은 데이터가 저장
- Agg + MV를 사용, 필요에 따라 다양한 시간단위 구성 가능
- 이는 실시간 데이터로 일간, 주간 데이터생성 가능 (별도 추출 없이)
- 불필요한 데이터 양을 감소

```
SELECT
    hour,
    count(1)
FROM dlcs_stms_array_basic_20201016
GROUP BY hour
ORDER BY hour ASC
```

hour	count(1)
0	22921816
1	17247819
2	12874090
3	10190747
4	8898753
5	9436776
6	12591934
7	17673427
8	21452430
9	24315158
10	26113467
11	26664050
12	26043587
13	26611642
14	26591223
15	25212606
16	22161399
17	22095287
18	20649254
19	20154589
20	20544568
21	20756743
22	20819462
23	21090327

```
SELECT
    hour,
    count(1)
FROM damtlcs_stms_array_basic_20201016
GROUP BY hour
ORDER BY hour ASC
```

hour	count(1)
0	961864
1	779568
2	628030
3	522582
4	464152
5	469001
6	566955
7	735225
8	919891
9	1097656
10	1278921
11	1154569
12	1171909
13	1206029
14	1323409
15	1431163
16	918207
17	811386
18	762158
19	740282
20	964580
21	850156
22	983516
23	1460522

```
DESCRIBE TABLE dlcs_stms_array_basic_20201016
```

name	type	default_type
date	Date	
starttime	DateTime	
endtime	DateTime	
hour	UInt32	
minute	UInt32	
stname	String	
origin	String	
age	String	
gender	String	
os	String	
browser	String	
pv	UInt64	
users	Array(String)	

13 rows in set. Elapsed: 0.001 sec.

```
DESCRIBE TABLE damtlcs_stms_array_basic_20201016
```

name	type
date	Date
hour	UInt32
stname	String
origin	String
age	String
gender	String
os	String
browser	String
pv	AggregateFunction(sum, Float64)
uv	AggregateFunction(uniq, String)

10 rows in set. Elapsed: 0.001 sec.

4.5 Clickhouse's Indexing (1/2)

DEVIEW 2020

Sorting Key(Primary Key) 를 이용한 최적화

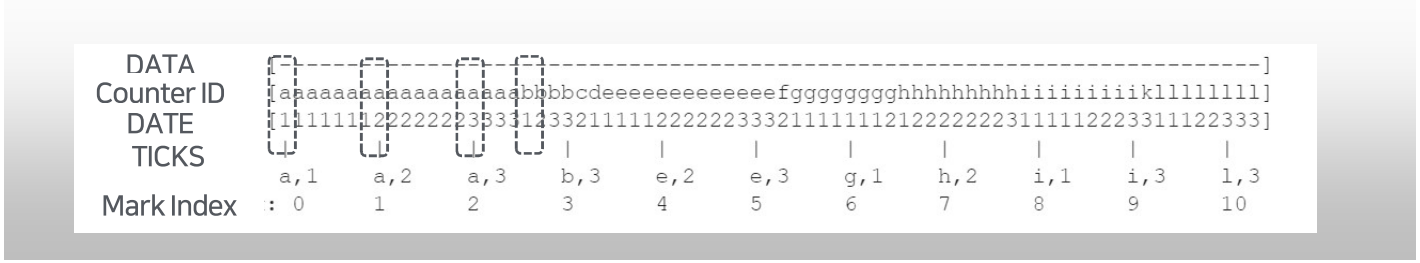
- Select Query 성능에 가장 기본적이며 중요한 설정
- 테이블 생성시 지정, 매우 효과적인 성능 개선가능

Creating a Table

```
CREATE TABLE [IF NOT EXISTS] [db.]table_name [ON CLUSTER cluster]
(
    name1 [type1] [DEFAULT|MATERIALIZED|ALIAS expr1] [TTL expr1],
    name2 [type2] [DEFAULT|MATERIALIZED|ALIAS expr2] [TTL expr2],
    ...
    INDEX index_name1 expr1 TYPE type1(...) GRANULARITY value1,
    INDEX index_name2 expr2 TYPE type2(...) GRANULARITY value2
) ENGINE = MergeTree()
ORDER BY expr
[PARTITION BY expr]
[PRIMARY KEY expr]
[SAMPLE BY expr]
[TTL expr [DELETE|TO DISK 'xxx'|TO VOLUME 'xxx'], ...]
[SETTINGS name=value, ...]
```

Clickhouse Index와 Granularity.

- select CounterID IN ('a', 'h') ➔ [0, 3) and [6, 8) Read
- select `CounterID IN ('a', 'h') AND Date = 3` ➔ [1, 3) and [7, 8) 을 Read
- Cardinality 가 높고 Frequent 할수록 유리 (1 Mark for 8192 Rows, Default)
- seconds 로 설정



4.5 Clickhouse's Indexing (2/2)

DEVIEW 2020

너무 긴 Primary Key의 수는 시스템 부하를 증가

- Insert 시 Sorting Key의 수를 증가
- Write Performance 를 하락
- Index Cache 가 사용하는 메모리를 증가 시킴

Left-Prefix Rule, Primary Key 순서가 성능에 민감한 영향

- Ex) (KEY1,KEY2) ➔ (KEY1KEY2) 상태로 Mark 와 strfind()

```
CREATE TABLE default.lcs_stms_array_basic_20190915
(
    `date` Date,
    `starttime` DateTime,
    `endtime` DateTime,
    `hour` UInt32,
    `minute` UInt32,
    `stname` String,
    `origin` String,
    `age` String,
    `gender` String,
    `os` String,
    `browser` String,
    `pv` UInt64,
    `users` Array(String)
)
ENGINE = MergeTree()
PARTITION BY toYYYYMM(date)
ORDER BY ${KEYS}
SETTINGS index_granularity = 8192
```

1st case : KEY="(stname, starttime)"
2nd case: KEY="(starttime, stname)"

```
SELECT
    hour,
    toUInt64(sum(pv / length(users))) AS pv,
    uniq(arrayJoin(users))
FROM default.lcs_stms_array_basic_20190915
WHERE stname LIKE 'SERVICE1/SERVICE2/SERVICE3%'
GROUP BY hour
ORDER BY hour ASC
```

1st case : KEY="(stname, starttime)"
#24 rows in set. Elapsed: 0.359 sec. **Processed 1.65 million rows**, 450.02 MB (4.61 million rows/s., 1.25 GB/s.)

2nd case KEY="(starttime, stname)"
#24 rows in set. Elapsed: 0.336 sec. **Processed 2.15 million rows**, 610.71 MB (6.39 million rows/s., 1.82 GB/s.)

4.6 Table Merge 를 고려한 Write

DEVIEW 2020

Background Merge 작업이 중요

- 짧은 단위로 자주 Writing 하는 것은 Write를 비효율적으로 만듦
- Background Merge 작업이 발생할 시간이 감소
- AMT 테이블도 데이터 생성에 지연 발생

Merge 작업을 고려한 Write Parameter 조절

- Write 단위는 크게 할 수록 유리 (Buffer Size)
- 각각의 Write 마다 Interval 은 충분히
- Merge 잘 일어날 수 있도록 Writing

- https://github.com/ivi-ru/flink-clickhouse-sink
- 가볍고 빠른 Clickhouse Sink

README.md

Flink-Clickhouse-Sink

build passing | maven central 1.1.0

Description

Flink sink for Clickhouse database. Powered by Async Http Client.

High-performance library for loading data to Clickhouse.

It has two triggers for loading data: *by timeout* and *by buffer size*.

Version map

flink	flink-clickhouse-sink
1.3.*	1.0.0
1.9.0	1.1.0

Install

Maven Central

```
<dependency>
  <groupId>ru.ivi.opensource</groupId>
  <artifactId>flink-clickhouse-sink</artifactId>
  <version>1.1.0</version>
</dependency>
```

```
globalParameters.put(ClickhouseSinkConsts.TIMEOUT_SEC, "5" );
```

```
propsURL.put(ClickhouseSinkConsts.MAX_BUFFER_SIZE, "5000");
```

4.7 H/A 구성 및 Monitoring Points (1/2)

DEVIEW 2020

Flink H/A 구성

- Checkpoint 구성 및 AutoRestart 구성
- Flink Cluster/Yarn 구성

Clickhouse H/A 구성

- Zookeeper 를 통한 Replication
- Circular Topology등 다양한 Replication 구성 가능
- Multi-Master Table , Distribution시 Master Preference 설정가능

4.7 H/A 구성 및 Monitoring Points (2/2)

DEVIEW 2020

Data Loss

- Flink 에서 감지되는 Late Data처리
- AllowedLateness 의 Updatable Store 가 지원이 되어야 함
- 그렇지 않은 경우 Drop 시키거나 별도로 처리하는 것이 더 효과적

> **Attention** You should be aware that the elements emitted by a late firing should be treated as updated results of a previous computation, i.e., your data stream will contain multiple results for the same computation. Depending on your application, you need to take these duplicated results into account or deduplicate them.

Duplicated Result 처리

- ReplacingMergeTree 사용가능
- Merge 전에 Read 가능성
- OPTIMIZE 자주 사용불가

```
CREATE TABLE default.url_pv
(
    `MetricDate` Date,
    `URL` String,
    `PV` UInt64,
    `Ver` UInt64
)
ENGINE = ReplacingMergeTree(Ver)
ORDER BY (MetricDate, URL)
```

```
insert into url_pv values ('2020-01-01', 'https://m.naver.com', 1, 1 );

insert into url_pv values ('2020-01-01', 'https://m.naver.com', 1, 2 );

insert into url_pv values ('2020-01-01', 'https://m.naver.com', 1, 3 );
```

```
SELECT *
FROM url_pv
```

MetricDate	URL	PV	Ver
2020-01-01	https://m.naver.com	1	3

MetricDate	URL	PV	Ver
2020-01-01	https://m.naver.com	1	1

MetricDate	URL	PV	Ver
2020-01-01	https://m.naver.com	1	2

```
OPTIMIZE TABLE url_pv FINAL
```

```
SELECT *
FROM url_pv
```

MetricDate	URL	PV	Ver
2020-01-01	https://m.naver.com	1	3

4.8 실시간 Backend 의 구성

실시간 Backend 의 구성

- Data Pipeline 을 거치며 Rows수를 효과적으로 줄이기 위해 노력
- 35 억 개 -> 5억 5천 (ArrayJoin)

- 조회 단위를 고려한 Materialized View + AggregatingMergeTree
- 5억 5천 Records 를 다시 2천5백만 Records 로 줄임
- 비약적인 성능 개선이 가능

- Source Table + Materialized View 의 조합은 매우 매력적
- Speed Layer의 데이터가 Batch Layer 에 데이터를 생성하는 셈

DEVIEW 2020

5. 효율적인 Vizualization

EFFICIENT

5.1 BI 도구의 필요성과 활용

DEVIEW 2020

Back-end API 서버 개발을 아우르며,
Front-end 차트 라이브러리까지 대체 가능한 시각화 제공

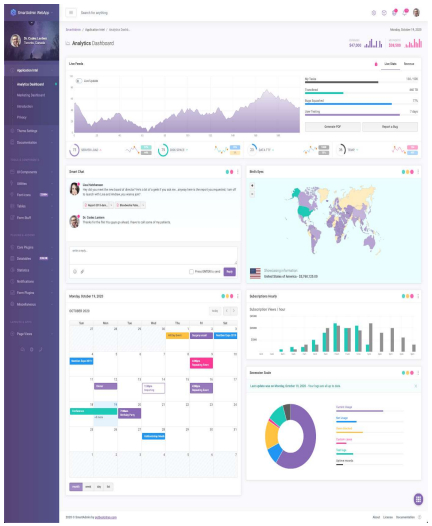


Chart X API

Chart Y, Z API

Chart α, β API

...

1 Data-source
in BI Tools

Data-Driven Documents

Google Developers Charts

Chart.js

5.1 BI 도구의 필요성과 활용

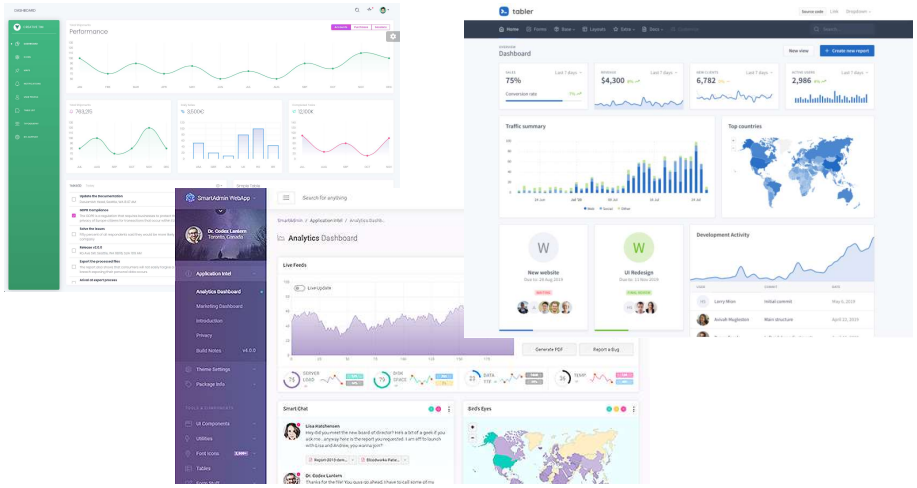
DEVIEW 2020

Back-end and Data engineer

Front-end of High productivity

Make possible

Web dashboard template



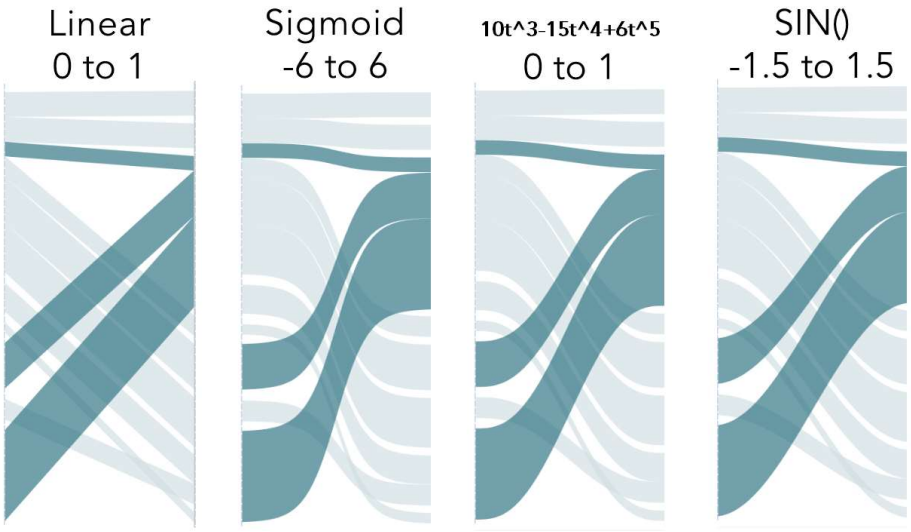
BI Tools for embedding

QlikView, Qlik Sense, Apache Superset, DOMO, tableau, talend, kibana, LinceBI, Power BI, KNIME, TIBCO Jaspersoft, Spotfire, pentaho, SAIKU, ORACLE, IBM Cognos Analytics, jedox, MicroStrategy, BIRT, SAP BusinessObjects, sas, spagobi, SEAL REPORT

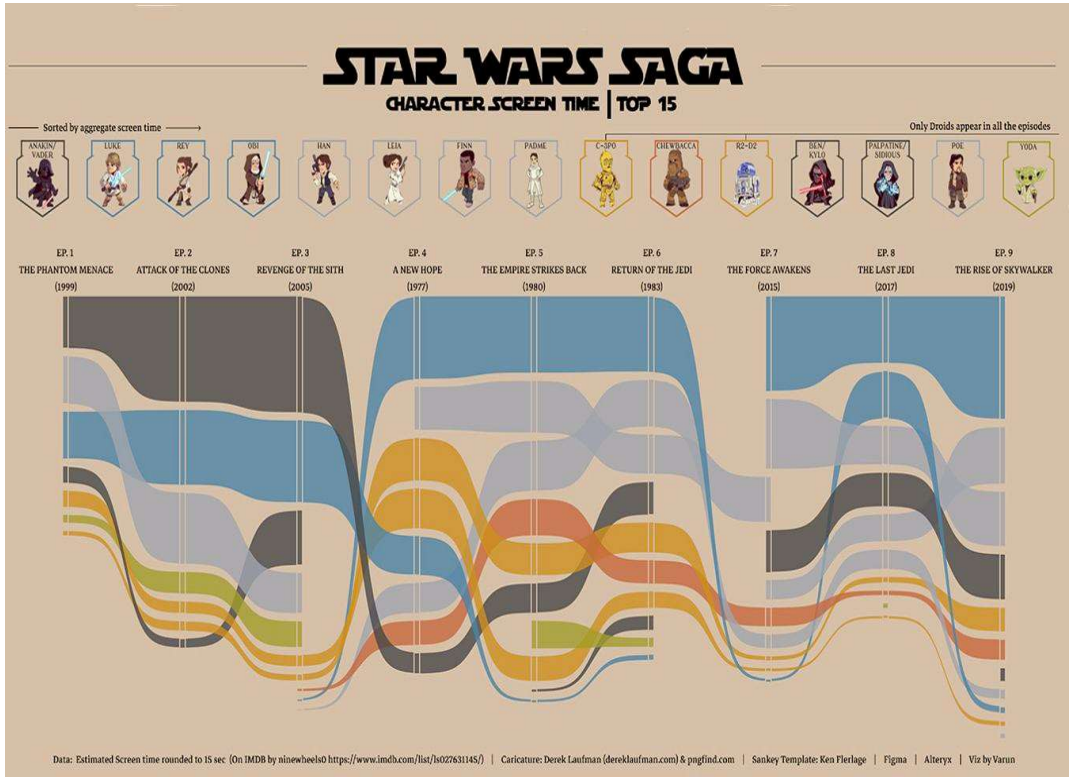
5.1 BI 도구의 필요성과 활용

DEVIEW 2020

시각화 자유도



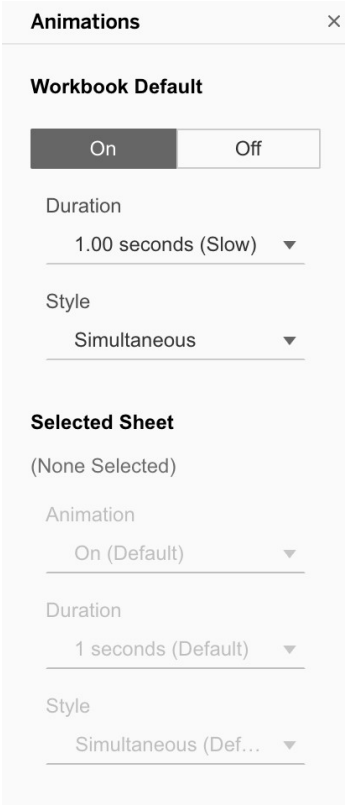
참고 - https://www.dataplusscience.com/Sigmoid.html



5.2 웹 템플릿 Embedding

DEVIEW 2020

실시간 차트 구현을 위한 애니메이션 및 윈도우 처리



Custom SQL Query Datasource

```
SELECT ...
FROM ...
WHERE timestamp BETWEEN
    <Parameters.window_start_time>
    AND
    addMinutes(<Parameters.window_start_time>, <Parameters.window_size_minute>)
```

Embedding with JS API – Time window parameters and forced CSR option

```
const containerDiv = document.getElementById(vizContainerId),
      url = "http://[TableauServerURL]/trusted/[AuthCode]/views/[WorkbookName]/[DashboardName]",
      options = {
        // only CSR
        ":render": "true",
        // window_start_time
        "[Parameters].[Parameter 1]": "2020-11-01 10:00:00",
        // window_size_minute
        "[Parameters].[Parameter 2]": 30,
        // window_current_time
        "[Parameters].[Parameter 3]": "2020-11-01 10:29:50"
      };
const viz = new tableau.Viz(containerDiv, url, options);
```

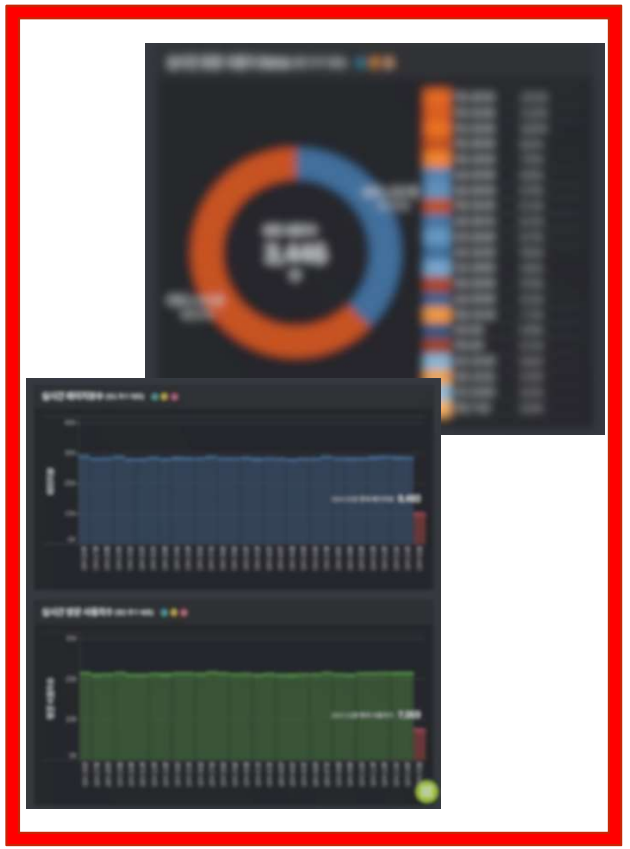
5.2 웹 템플릿 Embedding

DEVIEW 2020

실시간 차트 구현을 위한 애니메이션 및 윈도우 처리

Embedding with JS API – Time window processing

```
const
  options = {
    onFirstInteractive: function() {
      workbook = viz.getWorkbook();
      // 윈도우 슬라이딩 처리
      if (windowEndTime <= windowCurrentTime) {
        workbook.changeParameterValueAsync("window_start_time", windowStartTime).then(() => {
          viz.refreshDataAsync();
        })
      // 1 tick 이동 처리 (현재 시간의 위치 변경)
      } else {
        workbook.changeParameterValueAsync("window_current_time", windowCurrentTime);
      }
    }
  }
```



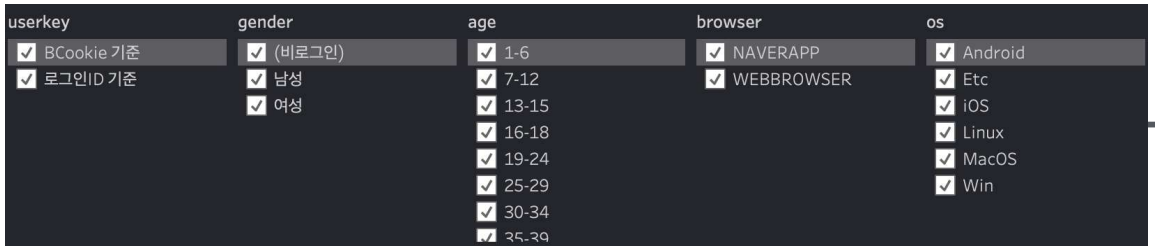
5.3 효과적인 UI를 위한 컴포넌트 개발

DEVIEW 2020

다차원 필터 연동과 연관(Relevant) 필터 처리의 구현

BI 도구의 필터 형식이 아닌 자체 구현 다차원 필터

BI 도구의 필터 형식의 다차원 필터



5.3 효과적인 UI를 위한 컴포넌트 개발

DEVIEW 2020

다차원 필터 연동과 연관(Relevant) 필터 처리의 구현

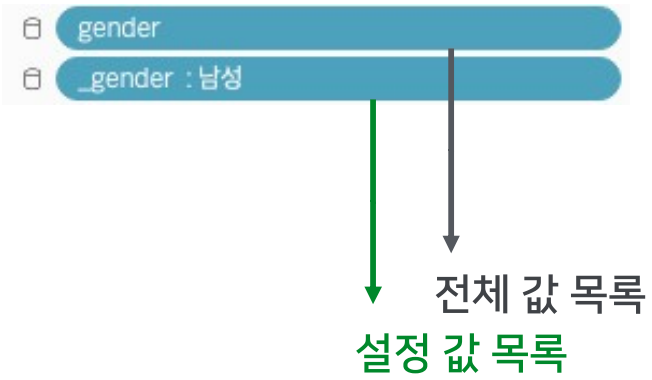
CategoricalFilter Class

Properties

Name	Type	Description
getIsExcludeMode()	bool	Gets a value indicating whether the filter is exclude or include (default).
getAppliedValues()	DataValue[]	Gets the collection of values that are currently set on the filter. This is a native JavaScript array and not a keyed collection. If more than 5000 values are selected, only the first 200 are returned.
getIsAllSelected()	bool	Gets a value indicating whether all values for the filter are selected. Returns true if all values in the filter are selected.

참고 - https://help.tableau.com/current/api/js_api/en-us/JavaScriptAPI/js_api_ref.htm#filtering

중첩 필터 구조



5.3 효과적인 UI를 위한 컴포넌트 개발

DEVIEW 2020

다차원 필터 연동과 연관(Relevant) 필터 처리의 구현

연관 필터가 아닌 경우의 필터

사용자가 OS 필터에서 “Windows” 선택

OS Version 필터의 값 목록

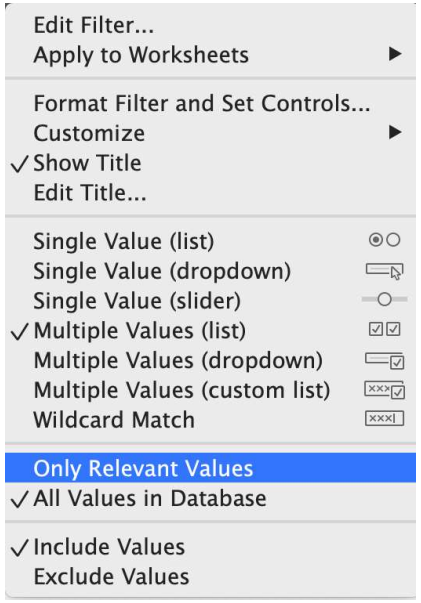
- Windows 7
- Windows XP
- Windows 10
- Catalina
- Mojave
- Sierra

연관 필터인 경우의 필터

사용자가 OS 필터에서 “Windows” 선택

OS Version 필터의 값 목록

- Windows 7
- Windows XP
- Windows 10
- ~~Catalina~~
- ~~Mojave~~
- ~~Sierra~~



5.3 효과적인 UI를 위한 컴포넌트 개발

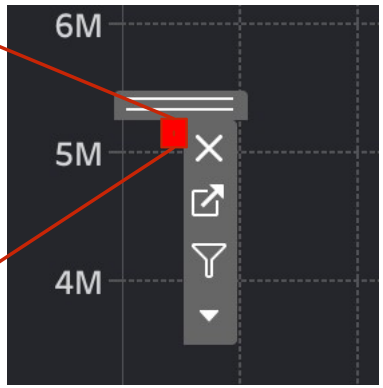
DEVIEW 2020

다차원 필터 연동과 연관(Relevant) 필터 처리의 구현

Filter values in worksheet

userkey	gender	age	browser	os	
로그인ID 기준	남성	1-6	NAVERAPP	Android	Abc
			WEBBROWSER	Android	Abc
		7-12	NAVERAPP	Android	Abc
				iOS	Abc
			WEBBROWSER	Android	Abc
				iOS	Abc
				Win	Abc
		13-15	NAVERAPP	Android	Abc
				iOS	Abc
			WEBBROWSER	Android	Abc
				iOS	Abc
				Linux	Abc
				MacOS	Abc
				Win	Abc

1x1 Worksheet in dashboard



5.3 효과적인 UI를 위한 컴포넌트 개발

DEVIEW 2020

다차원 필터 연동과 연관(Relevant) 필터 처리의 구현

Multi-Dimensional filter with relevant values

```
const dash = workbook.getActiveSheet();
const worksheetFilter = dash.getWorksheets().get("필터값 테이블의 워크시트명");

worksheetFilter.getFiltersAsync().then((filters) => {
  worksheetFilter.getSummaryDataAsync().then((dataTable) => {
    // 필터값 테이블에 존재하는 데이터만 현재 설정된 필터 값들로 처리
  });
});
```

```
// 사용자가 선택한 필터값을 워크시트들에 적용
curVizObjs.forEach((viz) => {
  const workbook = viz.getWorkbook();
  const worksheets = workbook.getActiveSheet().getWorksheets();
  worksheets.forEach((ws) => {
    ws.applyFilterAsync(
      // 워크북 필터링 설정을 위한 중첩 (field, _field) 필터링 적용된 경우,
      // _field 의 필터를 대신 설정해야함에 따른 처리
      isOverlapFilter ?
        '_' + filterName : filterName,
      // overlapFilter 일 경우 Alias 가 모두 해제되어 있음에 따른 분기
      isOverlapFilter ?
        appliedValues.map(obj => obj.id) :
        appliedValues.map(obj => obj.text),
      tableau.FilterUpdateType.REPLACE
    )
  });
});
```

5.3 효과적인 UI를 위한 컴포넌트 개발

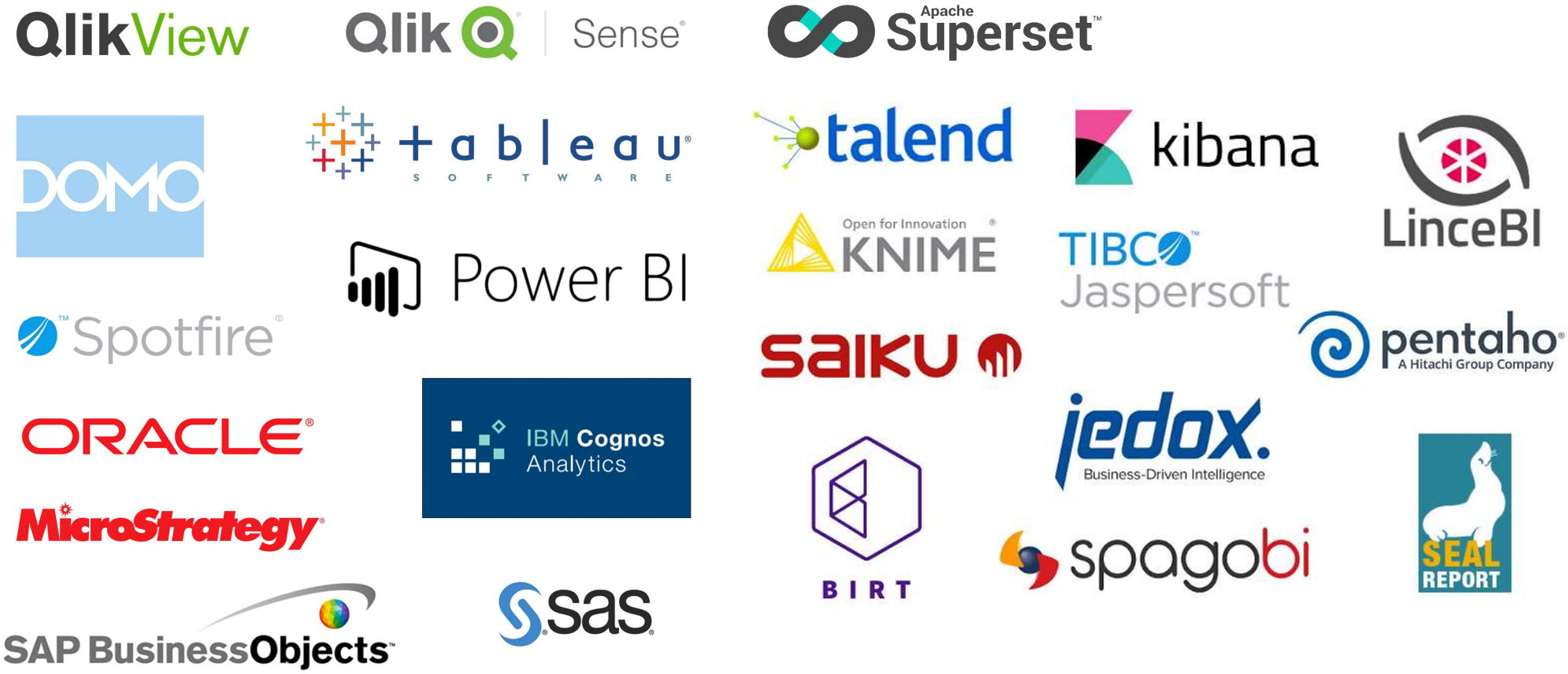
DEVIEW 2020

차트 없이, 데이터만으로 더 빠르고 유연하게

기존 차트 라이브러리의 활용

통계 데이터 다운로드 기능 제공

5.3 효과적인 UI를 위한 컴포넌트 개발



DEVIEW 2020

6. Wrap up

RESULT

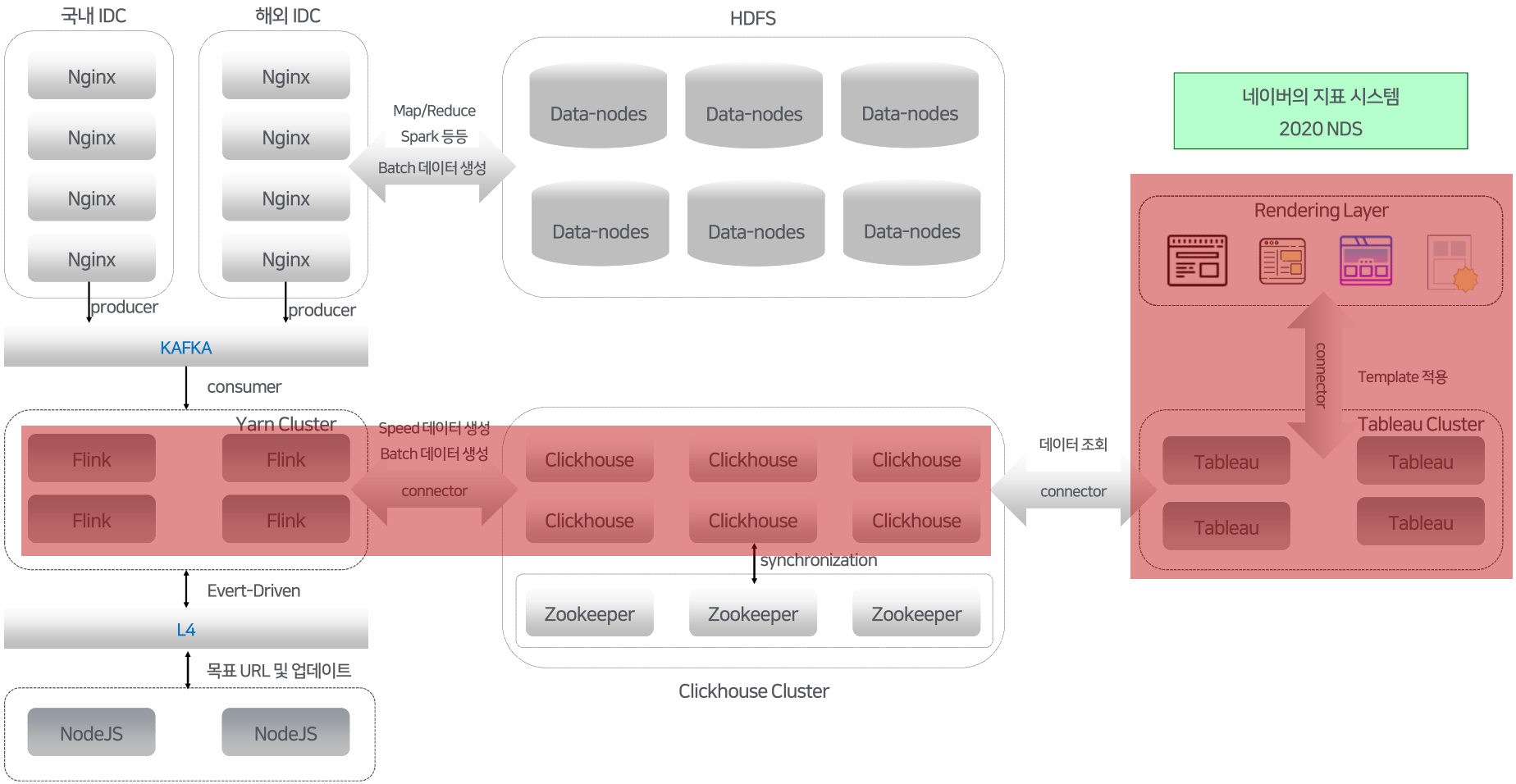
6.1 실시간 대시보드의 구성

DEVIEW 2020

1. 10초 단위의 실시간 지표
2. 어제 시간대별 지표와
   금일의 시간대별 지표 비교
3. 어제 시간대별 누적 그래프와
   금일의 실시간 누적 그래프 비교
4. 가장 작은 단위인 페이지URL 단위까지.
5. 사용자 데모 정보별, UA 별
   실시간 지표
6. 사용자 데모정보 별로 누적그래프와
   표로 중요한 dimension은 상세하게

• • •

6.2 전체적인 실시간 시스템 아키텍처



Q & A

Thank You