



**BIGDATA  
TEAM**



# Hive. SQL over Big Data

**Dral Alexey**, [aadral@bigdatateam.org](mailto:aadral@bigdatateam.org)

CEO at BigData Team, <http://bigdatateam.org/>

<https://www.facebook.com/bigdatateam/>

08.07.2019, Moscow, Russia



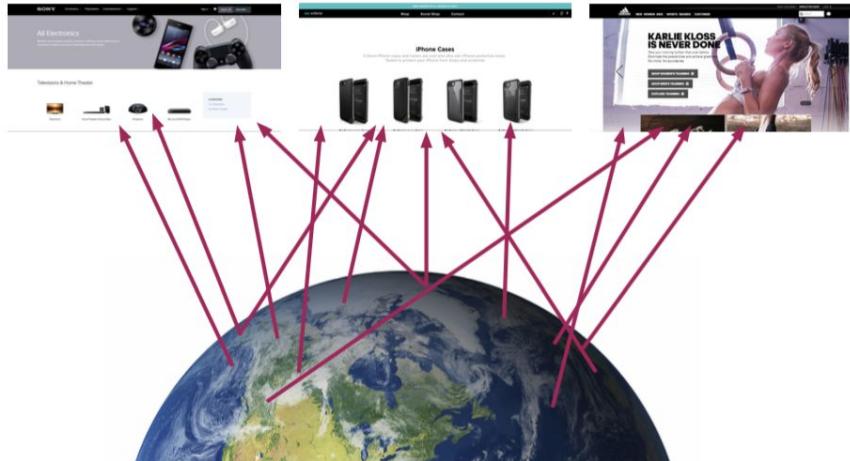
- ▶ [18:30 - 19:20] Мотивация Big Data SQL, Map-Side Join
  - перерыв, Q&A
- ▶ [19:30 - 20:10] Reduce-Side Join + оптимизации
  - перерыв, Q&A
- ▶ [20:20 - 21:30] Hive Workshop
  - ▷ External / Managed таблицы в Hive + RegexpSerDe
  - ▷ \*Солим косые данные (Data Skew)



# Мотивация Big Data SQL



# 1. Наиболее популярные регионы



109.188.67.224, Moscow City Centre, Moscow, Russia, Earth

109.188.67.221, Moscow City Centre, Moscow, Russia, Earth

...

Web-service  
access logs

Geobase



# 1. Наиболее популярные регионы



Web-service access logs



Geobase

format: ip, request, status\_code, ...

format: ip, region<sub>city</sub>, region<sub>country</sub>, ...

Join(ip)

+ WordCount(region) +

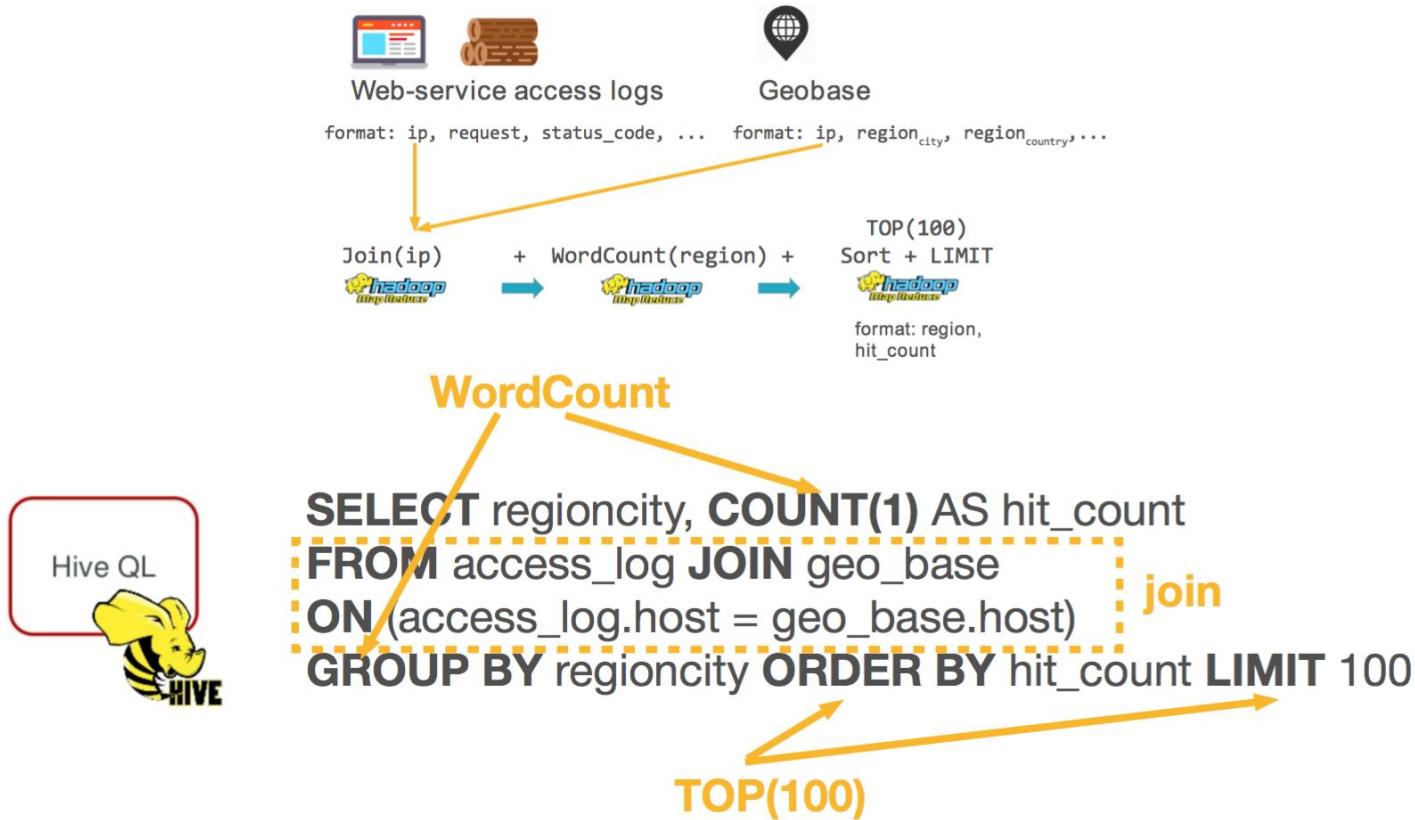
TOP(100)  
Sort + LIMIT



format: region,  
hit\_count

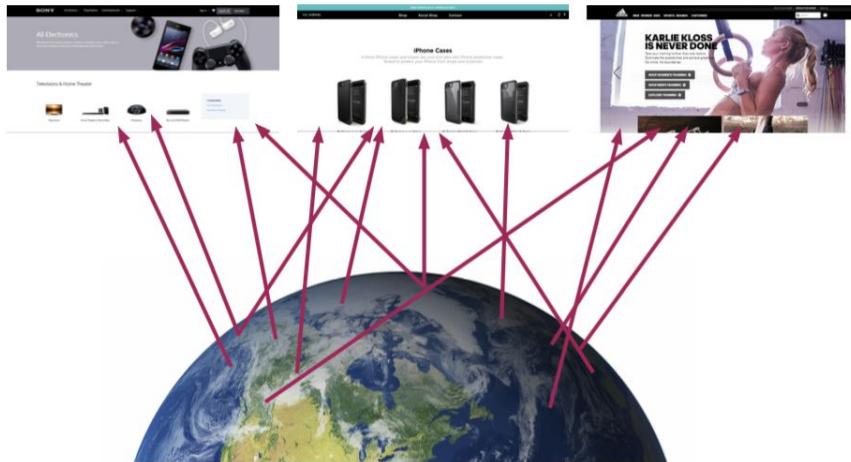


# 1. Наиболее популярные регионы





## 2. Доля роботных запросов



Web-service  
access logs

bot name + user\_agent<sub>1</sub> ip<sub>1</sub>  
bot name + user\_agent<sub>2</sub> + ip<sub>2</sub>  
...  
...



[ro]bot database



## 2. Доля роботных запросов



Web-service access logs



[ro]bot database

format: ip, request, user\_agent, ...

format: bot\_name, user\_agents, ips

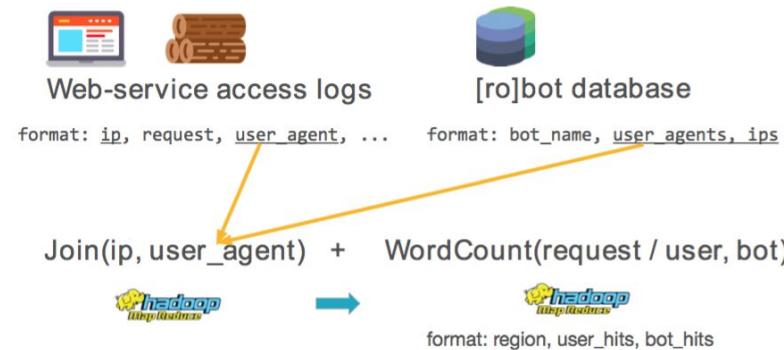
Join(ip, user\_agent) + WordCount(request / user, bot)



format: region, user\_hits, bot\_hits



## 2. Доля роботных запросов



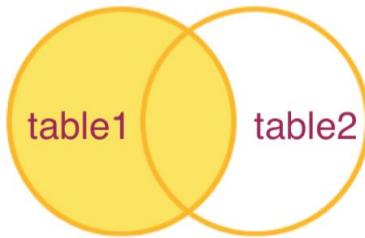
**Hive QL**

**HIVE**

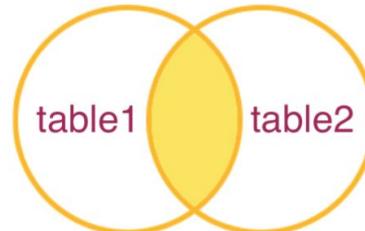
```
SELECT request,  
       SUM(IF(robot.bot_name IS NULL, 1, 0)) as user_hit_count,  
       SUM(IF(robot.bot_name IS NOT NULL, 1, 0)) as bot_hit_count  
FROM access_log LEFT OUTER JOIN robot ON (  
    access_log.host = robot.host  
    AND access_log.user_agent = robot.user_agent  
)  
GROUP BY request
```



LEFT JOIN

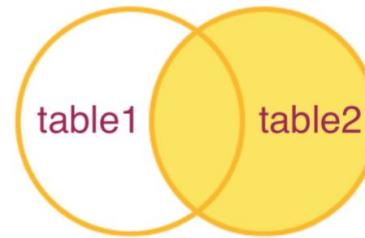


INNER JOIN

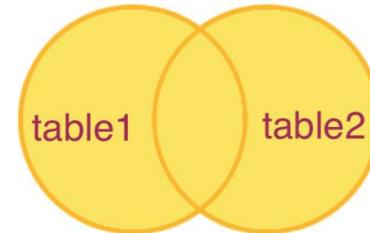


SQL Joins

RIGHT JOIN



FULL OUTER JOIN





### INNER JOIN

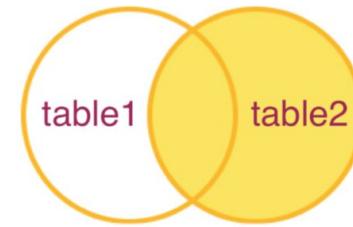
a	b
3	3
4	4

### LEFT JOIN

a	b
1	null
2	null
3	3
4	4

A	B
1	3
2	4
3	5
4	6

### RIGHT JOIN

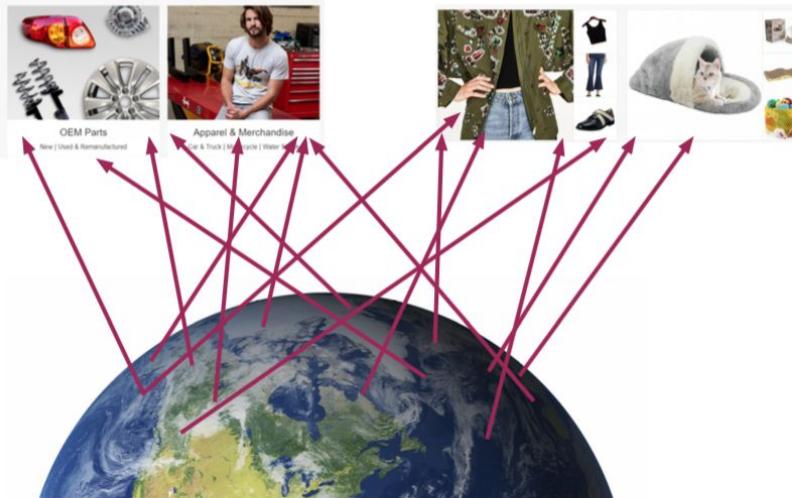


a	b
1	null
2	null
3	3
4	4
null	6
null	5

### FULL OUTER JOIN



### 3. Гендерное распределение



Web-service  
access logs



Geobase

	age	gender	occupation	zipcode
user_id				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213
6	42	M	executive	98101
7	57	M	administrator	91344
8	36	M	administrator	05201
9	29	M	student	01002
10	53	M	lawyer	90703



User personal data



### 3. Гендерное распределение в регионах



Web-service  
access logs



Geobase



User personal  
data

format: ip, request, user\_agent, ...      format: user\_id, gender, age, ...

format: ip, region<sub>city</sub>, region<sub>country</sub>, ...

Join(ip)

+

Join(user\_id)

+

WordCount(region/gender)

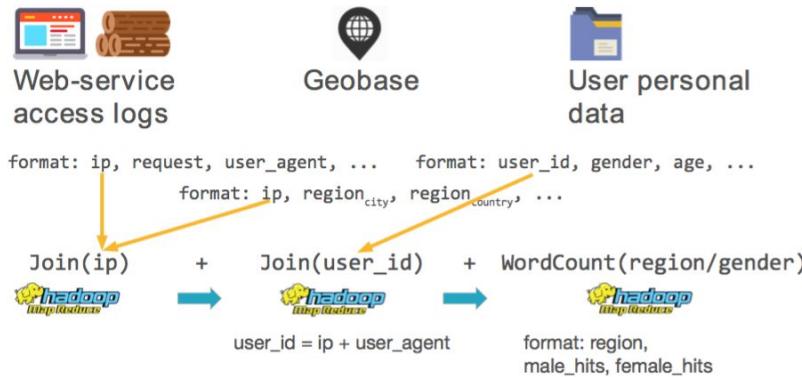


user\_id = ip + user\_agent

format: region,  
male\_hits, female\_hits



### 3. Гендерное распределение в регионах



```
SELECT regioncity,  
       SUM(IF(user.gender = "M",1,0)) as male_hit_count,  
       SUM(IF(user.gender = "F",1,0)) as female_hit_count
```

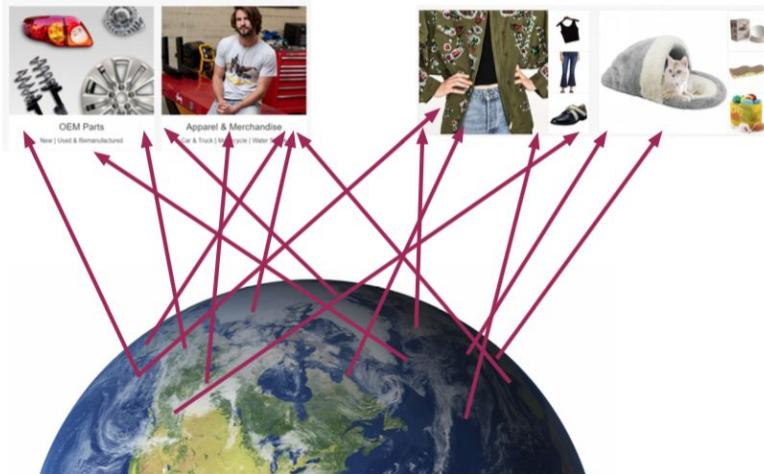
```
FROM access_log  
JOIN geo_base ON (access_log.host = geo_base.host)  
JOIN user ON (access_log.host = user.host  
             AND access_log.user_agent = user.user_agent  
)  
GROUP BY regioncity
```



two joins



## 4. Средний возраст клиента



Web-service  
access logs



Geobase

	age	gender	occupation	zipcode
user_id				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213
6	42	M	executive	98101
7	57	M	administrator	91344
8	36	M	administrator	05201
9	29	M	student	01002
10	53	M	lawyer	90703



User personal data



## 4. Средний возраст клиента



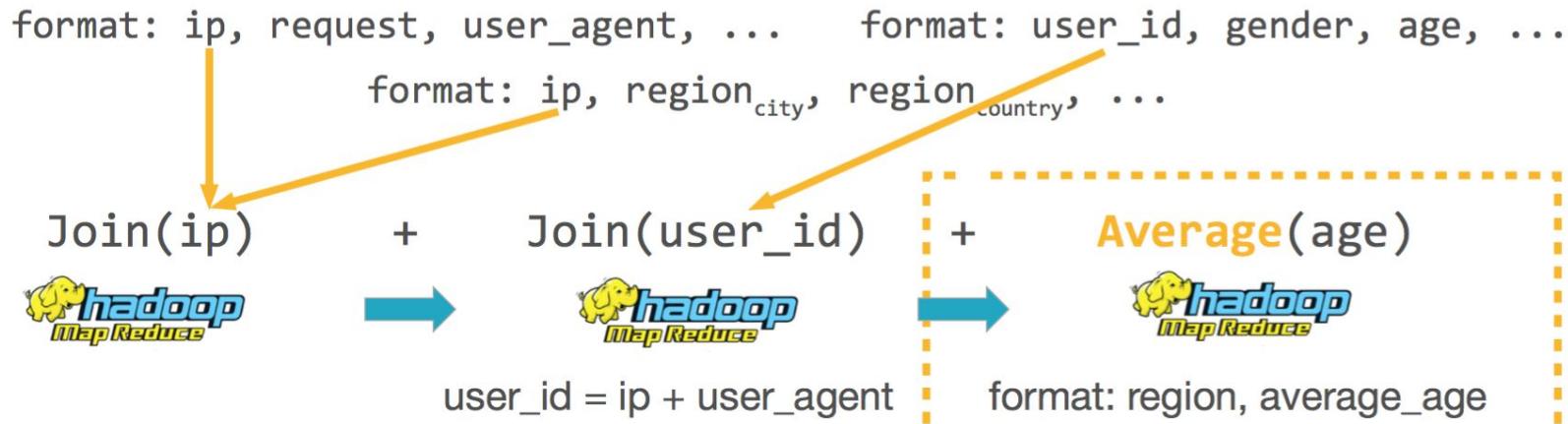
Web-service  
access logs



Geobase



User personal  
data





## 4. Средний возраст клиента



Web-service  
access logs



Geobase



User personal  
data

format: ip, request, user\_agent, ...      format: user\_id, gender, age, ...

format: ip, region<sub>city</sub>, region<sub>country</sub>, ...

Join(ip)

+

Join(user\_id)

hadoop  
Map Reduce



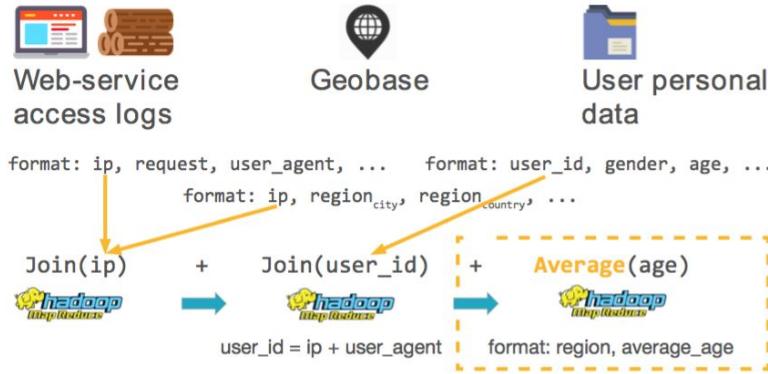
hadoop  
Map Reduce

user\_id = ip + user\_agent





## 4. Средний возраст клиента



Hive QL

**SELECT region2, AVG(user.age)**  
**FROM access\_log**  
**JOIN geo\_base ON (access\_log.host = geo\_base.host)**  
**JOIN user ON (access\_log.host = user.host**  
**AND access\_log.user\_agent = user.user\_agent**  
**)**  
**GROUP BY region2**

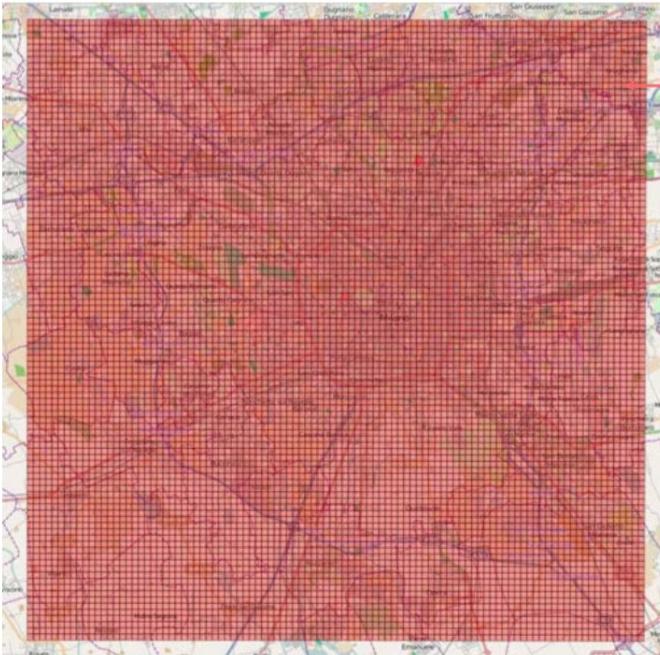
GenericUDFAverage  
Hive Java source code:  
<https://goo.gl/DCeVPk>



# Map- Side Join



# Telecommunications Dataset



Milano Grid

- **Square ID**
- › Time Interval
- › Country Code
- › SMS-in Activity
- › SMS-out Activity
- › Call-in Activity
- › Call-out Activity
- › Internet Traffic Activity

Schema

<https://dandelion.eu/datagems/SpazioDati/telecom-sms-call-internet-mi>



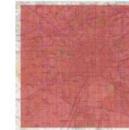
## BIG

- › Square ID
- › Time Interval
- › Country Code
- › SMS-in Activity
- › SMS-out Activity
- › Call-in Activity
- › Call-out Activity
- › Internet Traffic Activity

```
1 1383260400000 0 0.08136262351125882
1 1383260400000 39 0.14186425470242922
0.1567870050390246 0.16093793691701822
0.052274848528573205 11.028366381681026
1 1383261000000 0 0.13658782275823106
0.02730046487718618
1 1383261000000 33
0.026137424264286602
```

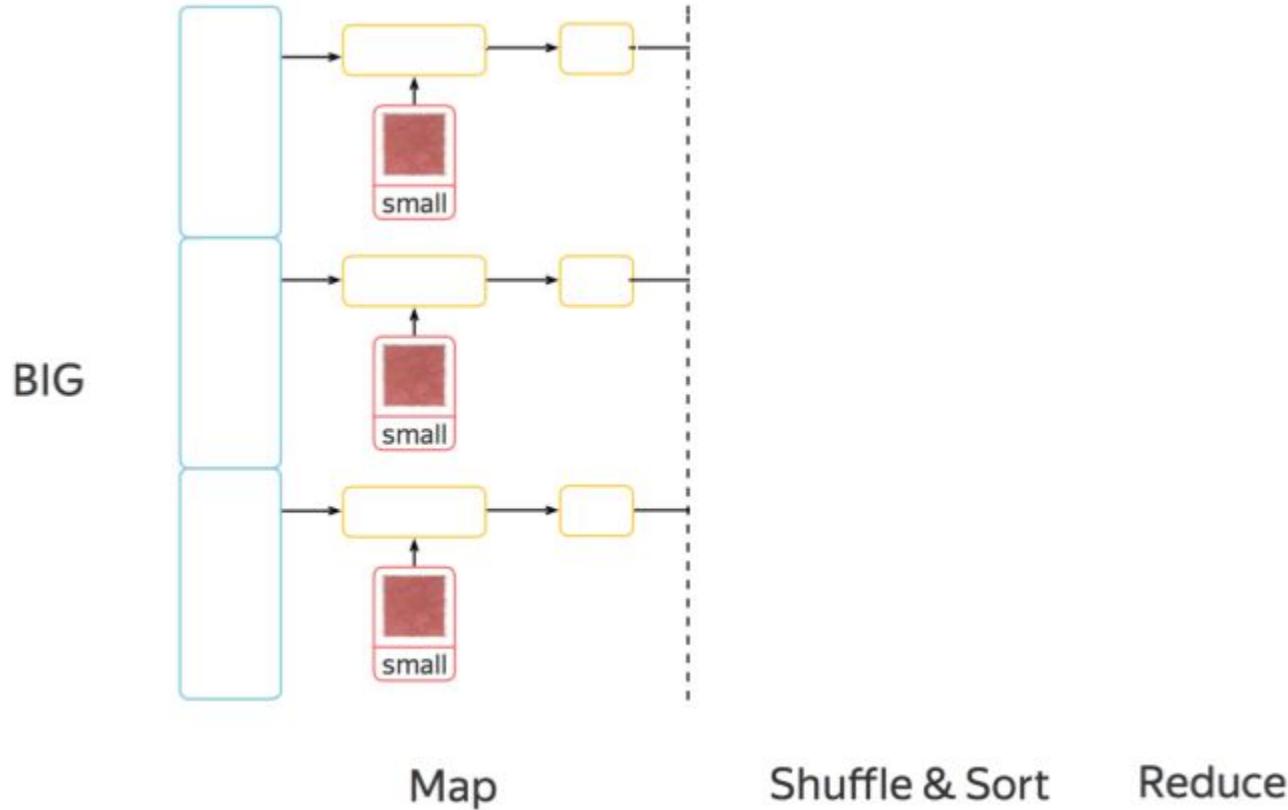
...

## small



```
{'type': 'Polygon', 'coordinates':
[[[9.0114910478323, 45.35880131440966],
[9.014491488013135, 45.35880097314403],
[9.0144909480813, 45.35668565341486],
[9.011490619692509,
45.356685994655464], [9.0114910478323,
45.35880131440966]]]}
```

...





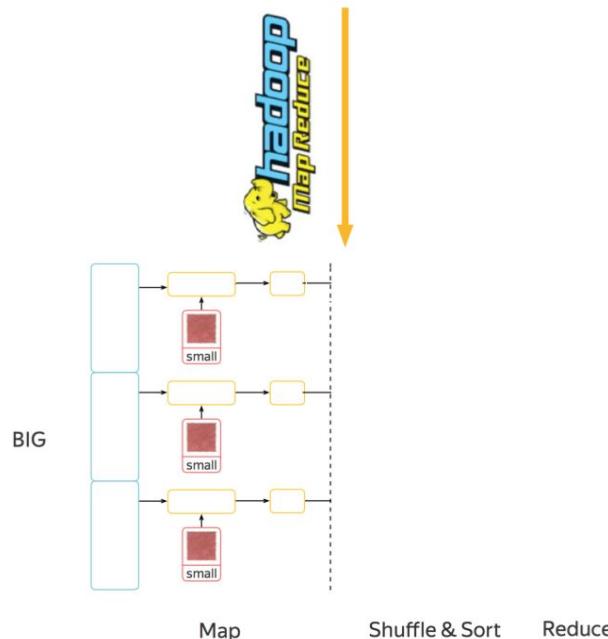
```
yarn jar $HADOOP_STREAMING_JAR \
  -files map_side_mapper.py,hdfs:///user/adral/milano-grid.geojson \
  -mapper 'python map_side_mapper.py' \
  -numReduceTasks 0 \
  -input /data/telecommunication \
  -output telecom-joins
```

**HDFS data  
Distributed Cache**





```
SELECT regioncity, COUNT(1) AS hit_count
FROM access_log JOIN geo_base
ON (access_log.host = geo_base.host)
GROUP BY regioncity ORDER BY hit_count LIMIT 100
```





```
yarn jar $HADOOP_STREAMING_JAR \
  -files map_side_mapper.py,hdfs:///user/adral/milano-grid.geojson \
  -mapper 'python map_side_mapper.py' \
  -numReduceTasks 0 \
  -input /data/telecommunication \
  -output telecom-joins
```

**HDFS data**  
**Distributed Cache**



1. Клиент: загружает датасет из HDFS
2. Строит hashtable
3. Загружает hashtable в Distributed Cache



**BIGDATA  
TEAM**

# Tea / Coffee Break



# Reduce- Side Join



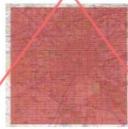
# Увеличение покрытия

**BIG**

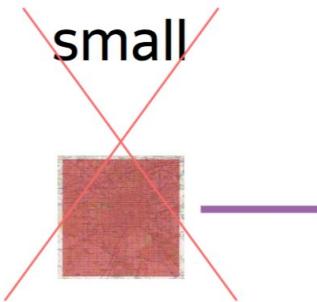
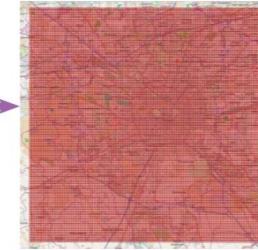
- › Square ID
- › Time Interval
- › Country Code
- › SMS-in Activity
- › SMS-out Activity
- › Call-in Activity
- › Call-out Activity
- › Internet Traffic Activity

```
1 1383260400000 0 0.08136262351125882
1 1383260400000 39 0.14186425470242922
0.1567870050390246 0.16093793691701822
0.052274848528573205 11.028366381681026
1 1383261000000 0 0.13658782275823106
0.02730046487718618
1 1383261000000 33
0.026137424264286602
...
...
```

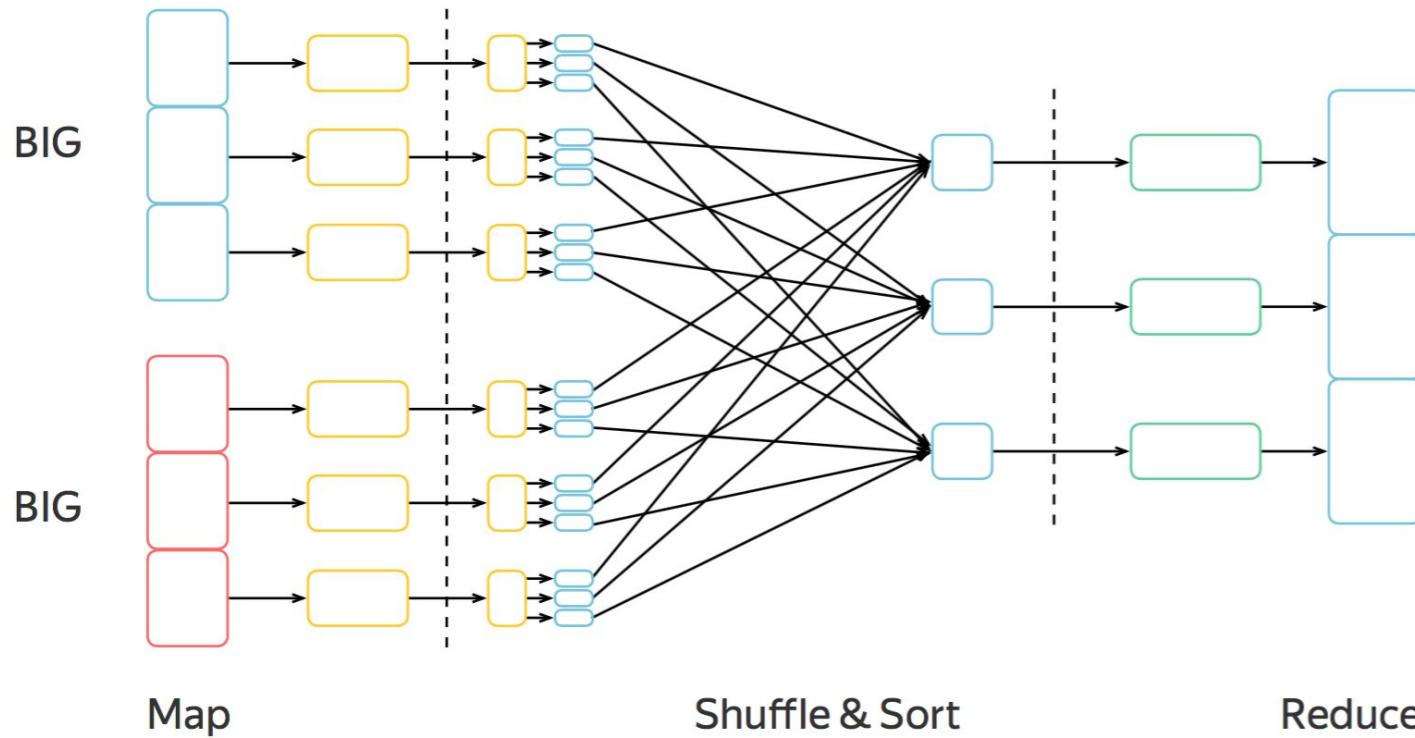
**small**



**BIG**



```
{"type": "Polygon", "coordinates": [
[[9.0114910478323, 45.35880131440966],
[9.014491488013135, 45.35880097314403],
[9.0144909480813, 45.35668565341486],
[9.011490619692509,
45.356685994655464], [9.0114910478323,
45.35880131440966]]]}
```





```
if "geojson" in os.environ["mapreduce_map_input_file"]:
    geojson = json.load(sys.stdin)
    grid = load_grid(geojson)
    for grid_id, ce11_type in grid.items():
        print(grid_id, "grid", ce11_type, sep="\t")
else
    for line in sys.stdin:
        square_id, aggregate = line.split("\t", 1)
        square_id = int(square_id)
        time_interval, country, sms_in, sms_out, call_in, call_out, internet = aggregate.split("\t"
        if sms_in:
            sms_in = float(sms_in)
            print(square_id, "logs", sms_in, sep="\t")
```



```
yarn jar $HADOOP_STREAMING_JAR \
    -files reduce_side_mapper.py \
    -mapper 'python reduce_side_mapper.py' \
    -numReduceTasks 0 \
    -input /data/telecommunication,/user/adral/geojson \
    -output telecom-joins
```

```
$ hdfs dfs -text telecom-joins/part-00010 | head -3
```

```
1 grid South ← string
2 grid South
3 grid South
```

```
$ hdfs dfs -text telecom-joins/part-00000 | head -3
```

```
1 logs 0.0813626235113 ← numeric
1 logs 0.141864254702
1 logs 0.136587822758
```



# Добавляем Identity Reducer

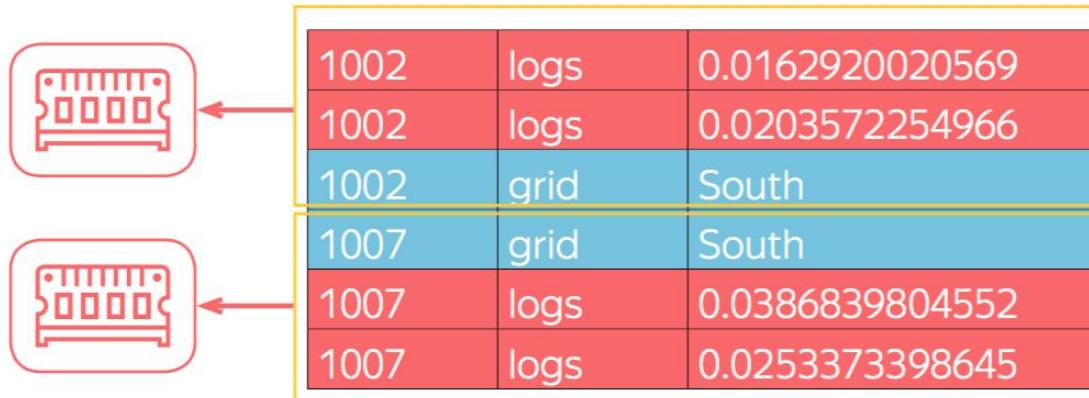
```
yarn jar $HADOOP_STREAMING_JAR \
-D mapreduce.partition.keypartitioner.options="-k1,1" \
-files reduce_side_mapper_slice.py \
-mapper 'python reduce_side_mapper.py' \
→ -numReduceTasks 5 \
-input /data/telecommunication,/user/adral/geojson \
-output telecom-joins \
-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

1002	logs	0.0162920020569
1002	logs	0.0203572254966
1002	grid	South
1007	grid	South
1007	logs	0.0386839804552
1007	logs	0.0253373398645



# Ограничения по памяти

```
yarn jar $HADOOP_STREAMING_JAR \
-D mapreduce.partition.keypartitioner.options="-k1,1" \
-files reduce_side_mapper_slice.py \
-mapper 'python reduce_side_mapper.py' \
-numReduceTasks 5 \
-input /data/telecommunication,/user/adral/geojson \
-output telecom-joins \
-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```



1002	logs	0.0162920020569
1002	logs	0.0203572254966
1002	grid	South
1007	grid	South
1007	logs	0.0386839804552
1007	logs	0.0253373398645



# Secondary Sort (via Comparator)

```
yarn jar $HADOOP_STREAMING_JAR \
-D mapreduce.partition.keypartitioner.options="-k1,1" \
-D stream.num.map.output.key.fields=2 \
-files reduce_side_mapper_slice.py \
-mapper 'python reduce_side_mapper_slice.py' \
-numReduceTasks 5 \
-input /data/telecommunication,/user/adral/geojson \
-output telecom-joins \
-partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

100	grid	South
100	logs	0.00422994505598
1002	grid	South
1002	logs	0.0241862339965
1007	grid	South
1007	logs	0.0145776778024
1011	grid	South
1011	logs	0.0627696965595
1016	grid	South
1016	logs	0.0123509364406

comparator



# Secondary Sort: Comparator Flags

```
yarn jar $SHADOOP_STREAMING_JAR \
  -D mapreduce.job.output.key.comparator.class=org.apache.hadoop.mapreduce.lib.partition.KeyFieldBasedComparator \
  -D mapreduce.partition.keycomparator.options="-k1,2r" \
  -D mapreduce.partition.keypartitioner.options="-k1,1" \
  -D stream.num.map.output.key.fields=2 \
  -files reduce_side_mapper_slice.py \
  -mapper 'python reduce_side_mapper_slice.py' \
  -numReduceTasks 5 \
  -input /data/telecommunication,/user/adral/geojson \
  -output telecom-joins \
  -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

9996	logs	0.149333295147
9996	grid	North
9991	logs	0.330465627227
9991	grid	North
9987	logs	0.0296826530265
9987	grid	North
9982	logs	0.262932749854
9982	grid	North
998	logs	0.0881801546604
998	grid	South



BIGDATA  
TEAM

Join Options



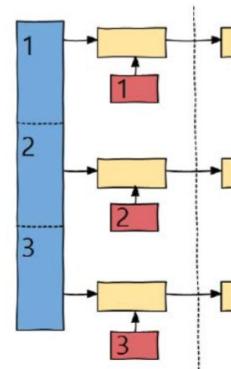


```
CREATE TABLE access_log (
    ...
)
CLUSTERED BY (ip)
    INTO 128 BUCKETS
...
;
```

# Bucket Map- Side Join

```
CREATE TABLE geo_base (
    ...
)
CLUSTERED BY (ip)
    INTO 128 BUCKETS
...
;
```

## Bucket Join



Map Phase



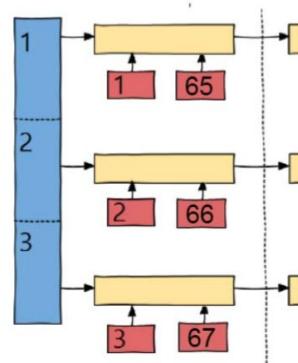
```
CREATE TABLE access_log (
    ...
)
CLUSTERED BY (ip)
    INTO 64 BUCKETS
...
;
```

# Bucket Map- Side Join

```
CREATE TABLE geo_base (
    ...
)
CLUSTERED BY (ip)
    INTO 128 BUCKETS
...
;
```

## Bucket Join

$\text{hash(ip)} \% 64 == 1$



$\text{hash(ip)} \% 128 == 1,65$

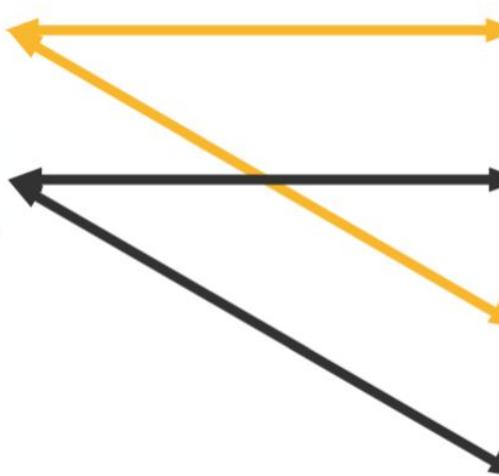
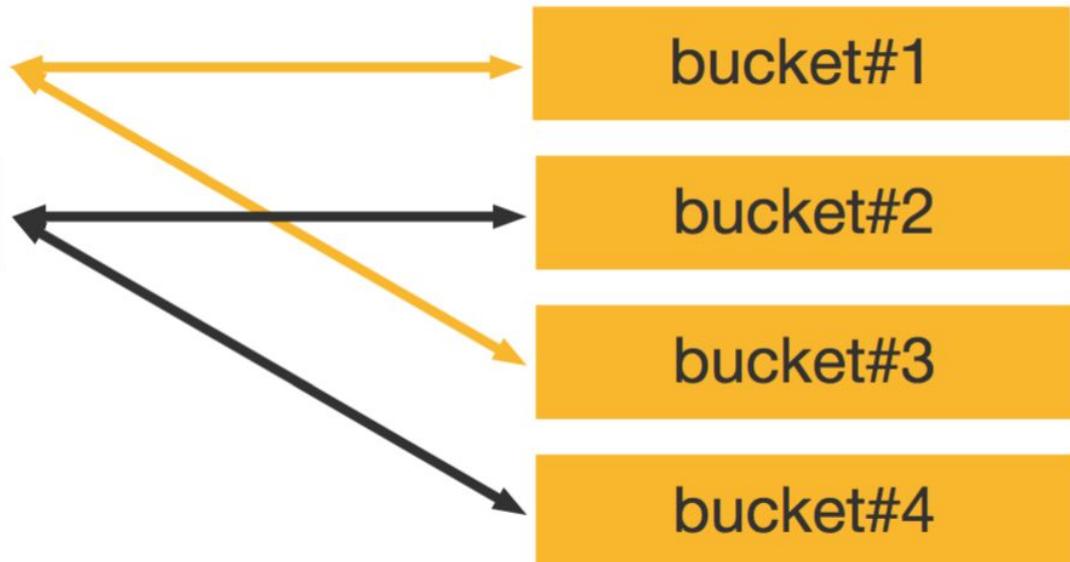
Map Phase



Table A



Table B



# Bucket Map- Side Join

Table A



Table B

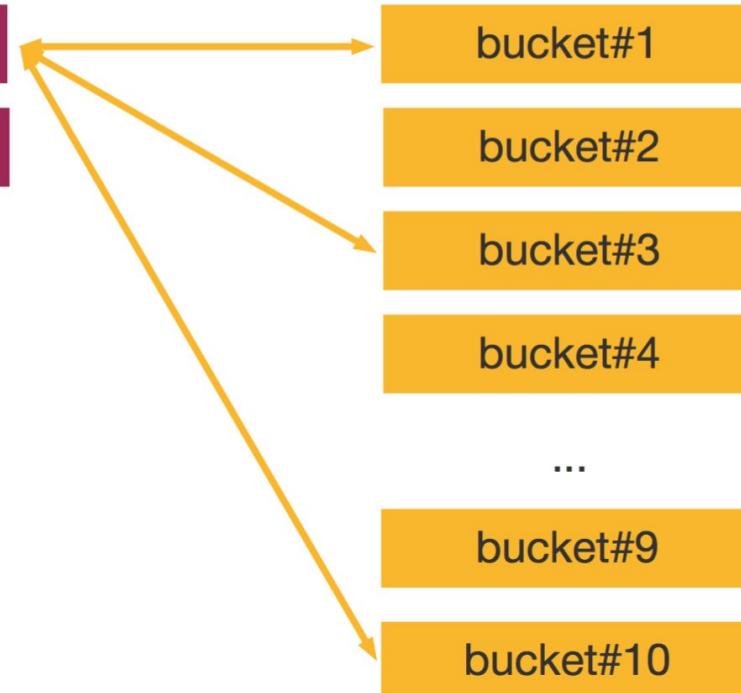




Table A

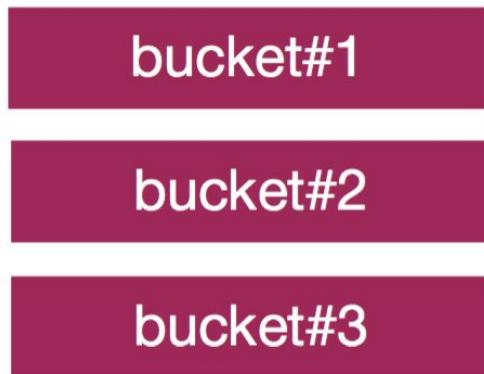
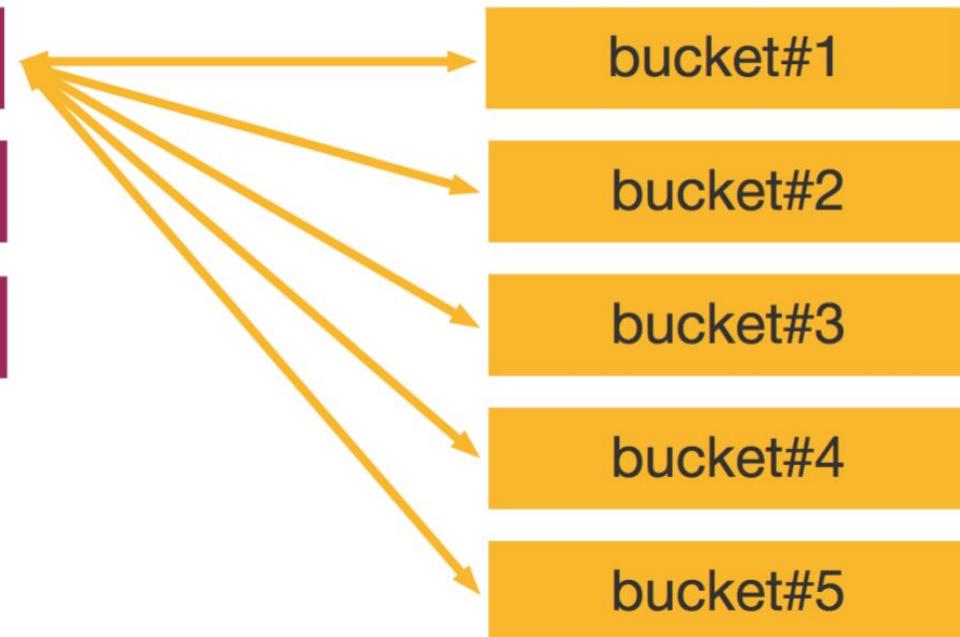
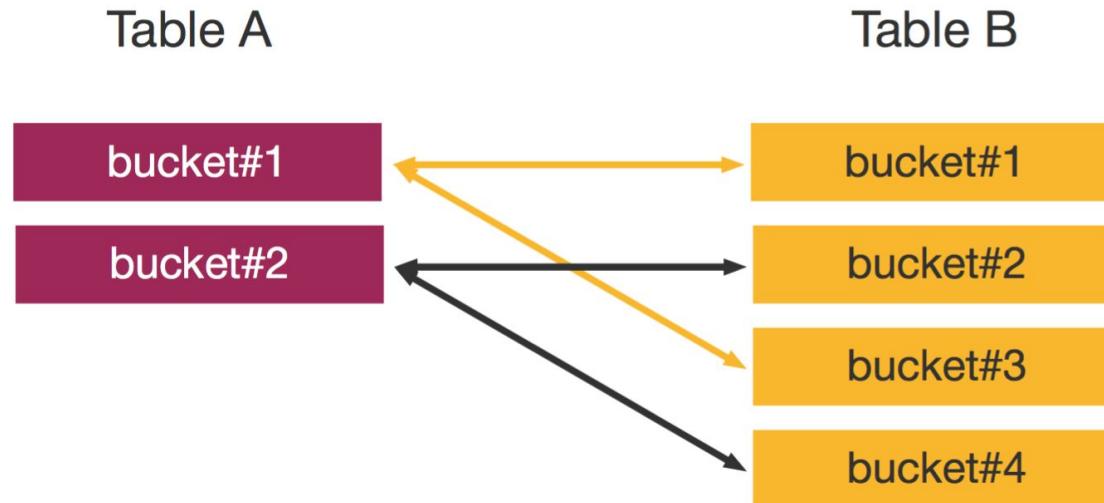


Table B



**СОВЕТ:** Использовать степени двойки

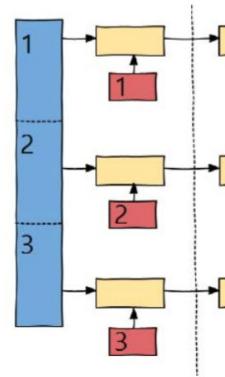


1. Если повезет, запустится Bucket Map-Side Join
2. Наиболее эффективно использует Namenode RAM<sub>42</sub>

# Оптимизация Bucket Map-Side Join

```
CREATE TABLE access_log (
    ...
)
CLUSTERED BY (ip)
    SORTED BY (ip)
    INTO 128 BUCKETS
...
;
```

## Bucket Join limitations?



Map Phase



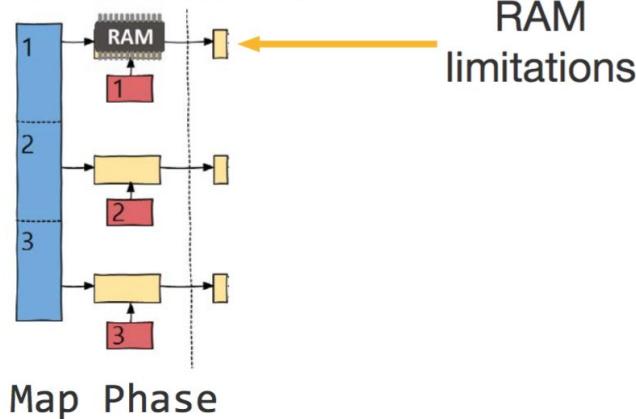
# Bucket Map- Side Join Optimization

```
CREATE TABLE access_log (
    ...
)
CLUSTERED BY (ip)
    SORTED BY (ip)
    INTO 128 BUCKETS
...
;
```

see: [https://en.wikipedia.org/wiki/Merge\\_sort](https://en.wikipedia.org/wiki/Merge_sort)  
see: [https://en.wikipedia.org/wiki/External\\_sorting](https://en.wikipedia.org/wiki/External_sorting)



## Bucket Join limitations?

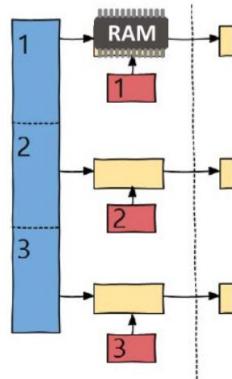




# Sort-Merge-Bucket Join

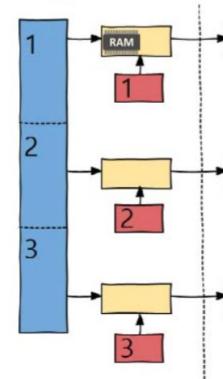
```
CREATE TABLE access_log (
    ...
)
CLUSTERED BY (ip)
    SORTED BY (ip)
    INTO 128 BUCKETS
...;
```

Bucket Join



Map Phase

Sort-Merge-Bucket Join



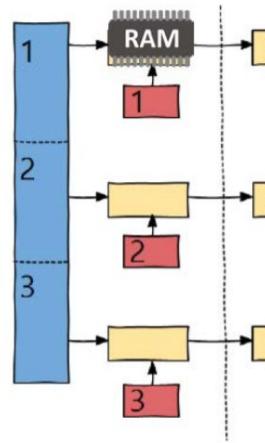
Map Phase

use: merge-sort



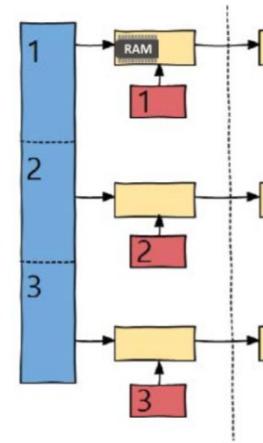
# Sort-Merge-Bucket Join

## Bucket Join



use: merge-sort

## Sort-Merge-Bucket Join



Map Phase

Map Phase

```
hive> SET hive.auto.convert.sortmerge.join=true;
```

...



# External / Managed таблицы в Hive + RegexpSerDe



KEEP  
CALM  
AND  
TRY  
CODING

-  Вы знаете, какие типы Join бывают в Hadoop / Hive (Map-, Reduce-, Bucket- и SMB).
-  Вы умеете (и умели еще с прошлого занятия) решать задачу вторичной сортировки (Secondary Sort)
-  Вы умеет пользоваться Managed и External таблицами без риска удаления production данных, а также RegexpSerDe для задания произвольной схемы.

# Thank you! Questions?

Feedback: [http://rebrand.ly/mf2019q2\\_feedback\\_04\\_hive](http://rebrand.ly/mf2019q2_feedback_04_hive)

**Dral Alexey**, [aadral@bigdatateam.org](mailto:aadral@bigdatateam.org)

CEO at BigData Team, <http://bigdatateam.org/>

<https://www.linkedin.com/in/alexey-dral>

<https://www.facebook.com/bigdatateam/>



# Appendix



# Материалы для погружения

-  Семплирование данных в Hive:  
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling>
-  Бакетированные данные в Hive:  
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL+BucketedTables>
-  UDF, UDAF, UDTF:  
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+UDF>
-  Работа с вложенными данными:  
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+LateralView>
-  Работа с View:  
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL#LanguageManualDDL-Create/Drop/AlterView>



# Материалы для погружения

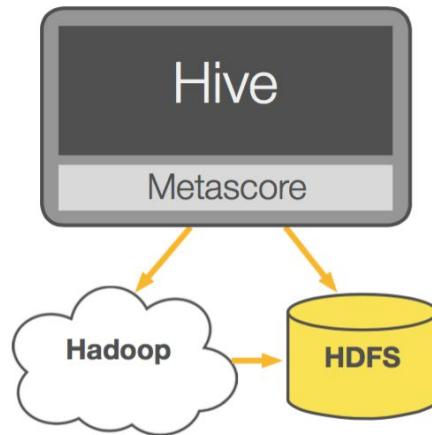
-  Настраиваем Hive таблицы поверх данных в HDFS:  
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL>
-  Пользуемся клаузой “EXPLAIN” для того, чтобы понять план выполнения Hive-запроса:  
<https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Explain>



# Hive DML



# Hive DML: Implementation Q&A



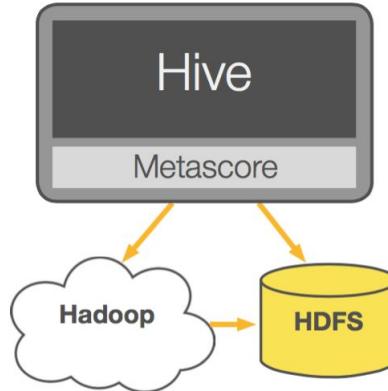
- (1) DDL
- (2) **HiveQL (details)**
- (3) DML

## MapReduce (?)

- SELECT .. FROM
- WHERE
- GROUP BY + HAVING
- JOIN
- ORDER BY / **SORT BY**



# Hive DML: Implementation Q&A



- (1) DDL
- (2) **HiveQL (details)**
- (3) DML

## MapReduce (?)

- SELECT .. FROM [<-- Map]
- WHERE [<-- Map]
- GROUP BY [<-- Shuffle & Sort] + HAVING [<-- Reduce]
- JOIN [<-- Map / Reduce "-side"]
- ORDER BY / SORT BY [<-- Reduce]





## EXPLAIN

```
FROM src
INSERT OVERWRITE TABLE dest_g1
SELECT src.key, sum(substr(src.value,4))
GROUP BY src.key;
```

### (1) The Abstract Syntax Tree

ABSTRACT SYNTAX TREE:

```
(TOK_QUERY (TOK_FROM (TOK_TABREF src)))
```

...

### (2) The Dependency Graph

STAGE DEPENDENCIES:

Stage-1 is a root stage

Stage-2 depends on stages: Stage-1

Stage-0 depends on stages: Stage-2

### (3) The plans of each Stage

STAGE PLANS:

Stage: Stage-1

Map Reduce

Alias -> Map Operator Tree:

src

Reduce Output Operator

key expressions:

expr: key

type: string

sort order: +