MySQL DBA解锁数据分析的新姿势 --ClickHouse

Power Your Data

新浪-高鹏-2017年12月

“工具选的好，下班回家早”

目录

- 数据分析工具的选择
- ClickHouse原理、架构
- ClickHouse在新浪的实践与经验
- ClickHouse案例、生态

关于我

我是谁?

我是干啥的?

关于我

DBA

关于我

DBA

DA

关于我们

Data Analyst
Data Translator

致力于运维大数据
挖掘与分析

可视化、报警、数据分析

AI OPS

“表哥”“表姐”们

我们需要什么样的工具?

Excel?

也用

Hadoop Spark Hive

?


But,
Hadoop这玩意，
不是一天就
能玩得转的啊~~

Google眼中的
Hadoop

多数人眼中的
Hadoop

太重了~

一切以需求作为第一位~

一切以需求作为第一位~

快速~ 好用~ 体量够用~

一切以需求作为第一位~

快速~ 好用~ 体量够用~

好维护！！

对结构化的数据

快速给出聚合/过滤结果

We Need

SQL

Fast SQL

Fast Complex
SQL

没有什么数据统计是一个SQL解决不了的。

如果有，那就2个

俄罗斯搜索巨头Yandex开源

异步复制 OLAP

SQL

最终一致

统计函数

压缩

PB级别

列式存储

集群

驱动丰富

updated in real time

超高性能

线性扩展

跨数据中心

然鹅，

不支持事务

不支持update/delete

But,

查询‘巨’快

超大容量

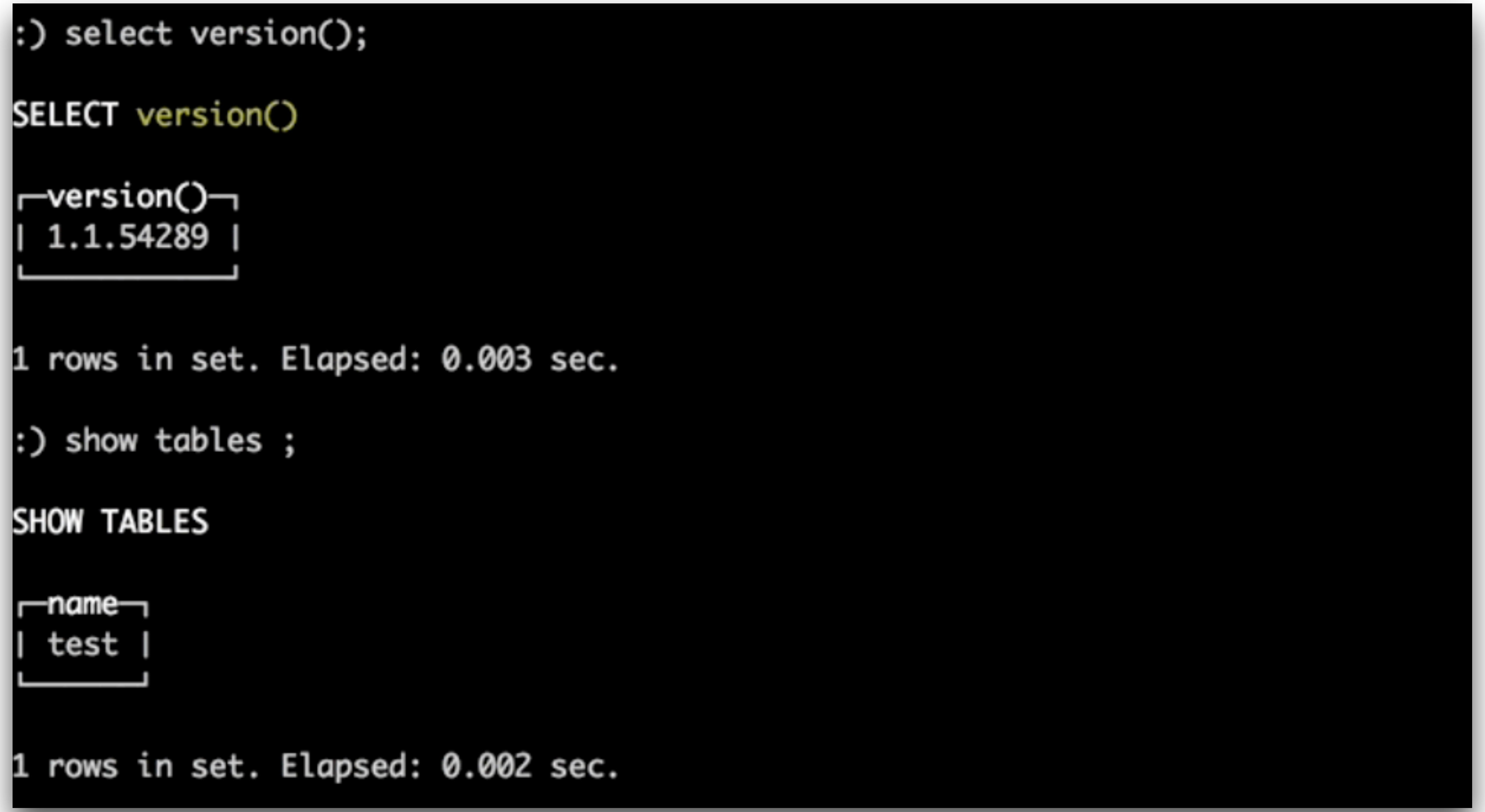
Let's Begin

部署： 单机

部署

1. 官方提供Ubuntu包
2. 第三方rpm包
2. Docker镜像

需要注意：
1. 修改网络，默认监控IPv4/v6
2. 自定义数据目录，修改官方启动脚本
3. Docker修改时区



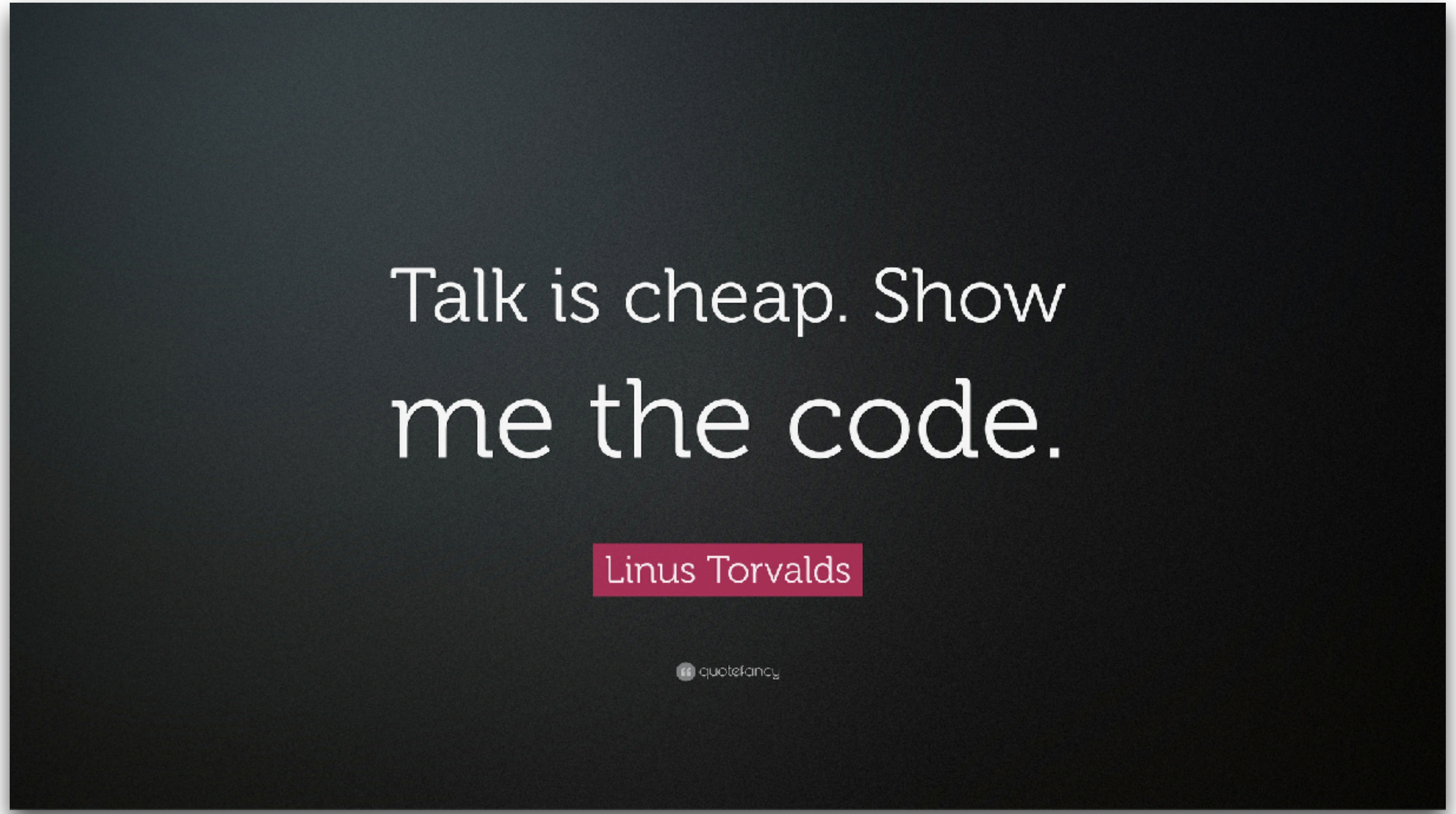
```
:) select version();

SELECT version()

┌─version()─┐
│ 1.1.54289 │
└───────────┘

1 rows in set. Elapsed: 0.003 sec.

:) show tables ;

SHOW TABLES

┌─name─┐
│ test │
└──────┘

1 rows in set. Elapsed: 0.002 sec.
```

部署： 单机

是不是很SQL

部署： 单机

蚝，
我们来压测一下~



部署： 单机

数据源

USA civil flights data
since 1987 till 2015

contains 166 millions rows
63 GB of uncompressed data

```
# 下载数据
for s in `seq 1987 2017`
do
for m in `seq 1 12`
do
wget http://transtats.bts.gov\
/PREZIP/On_Time_On_Time_Performance_${s}_${m}.zip
done
done

# 解压
for i in `ls *.zip`; do unzio -o $i;done

# 插入数据
for i in `ls *.csv`
do
echo '__________________'
echo $i
du -sh $i
wc -l $i
time cat $i | sed 's/\.00//g' | sed 1d | clickhouse-client \
-h  127.0.0.1 --port 9000  -d gaopeng4  \
--query="INSERT INTO ontime FORMAT CSVWithNames";
echo '__________________'
sleep 2
done
```

https://clickhouse.yandex/docs/en/getting_started/example_datasets/ontime.html

部署：单机

数据源

USA civil flights data
since 1987 till 2015

contains 166 millions rows
63 GB of uncompressed data

数据大小	173MB
文件行数	436951
插入耗时	4.731 Sec
平均速度	9.3 W/Sec
压缩率	5倍

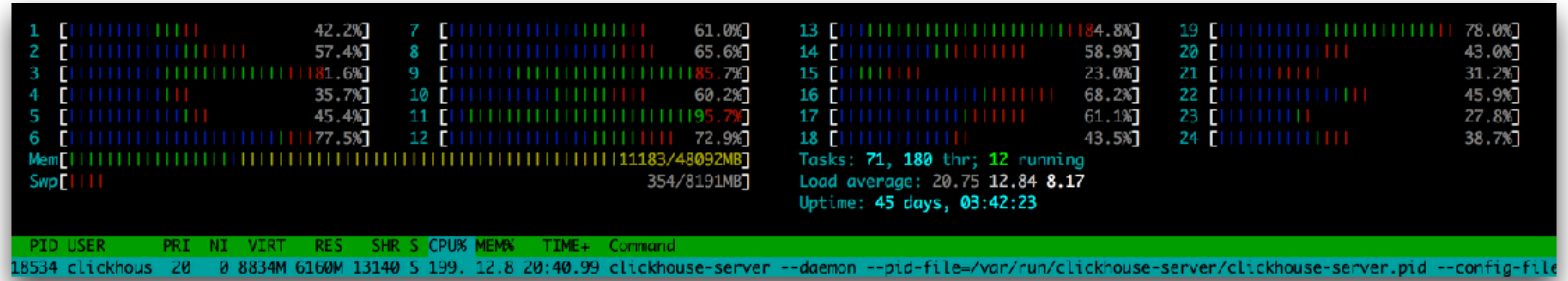
https://clickhouse.yandex/docs/en/getting_started/example_datasets/ontime.html

部署： 单机

并发5个进程

```
while read a b c d e ;
do
echo $a $b $c $d $e;

time cat $a | sed 's/\.00//g' | sed 1d | clickhouse-client -h 127.0.0.1 \
--port 9000 -d gaopeng4 --query="INSERT INTO ontime FORMAT CSVWithNames" &

time cat $b | sed 's/\.00//g' | sed 1d | clickhouse-client -h 127.0.0.1 \
--port 9000 -d gaopeng4 --query="INSERT INTO ontime FORMAT CSVWithNames" &

time cat $c | sed 's/\.00//g' | sed 1d | clickhouse-client -h 127.0.0.1 \
--port 9000 -d gaopeng4 --query="INSERT INTO ontime FORMAT CSVWithNames" &

time cat $d | sed 's/\.00//g' | sed 1d | clickhouse-client -h 127.0.0.1 \
--port 9000 -d gaopeng4 --query="INSERT INTO ontime FORMAT CSVWithNames" &

time cat $e | sed 's/\.00//g' | sed 1d | clickhouse-client -h 127.0.0.1 \
--port 9000 -d gaopeng4 --query="INSERT INTO ontime FORMAT CSVWithNames" &

sleep 5
done <<EOF
`ls *.csv | xargs -n 5`
EOF
```

机器负载

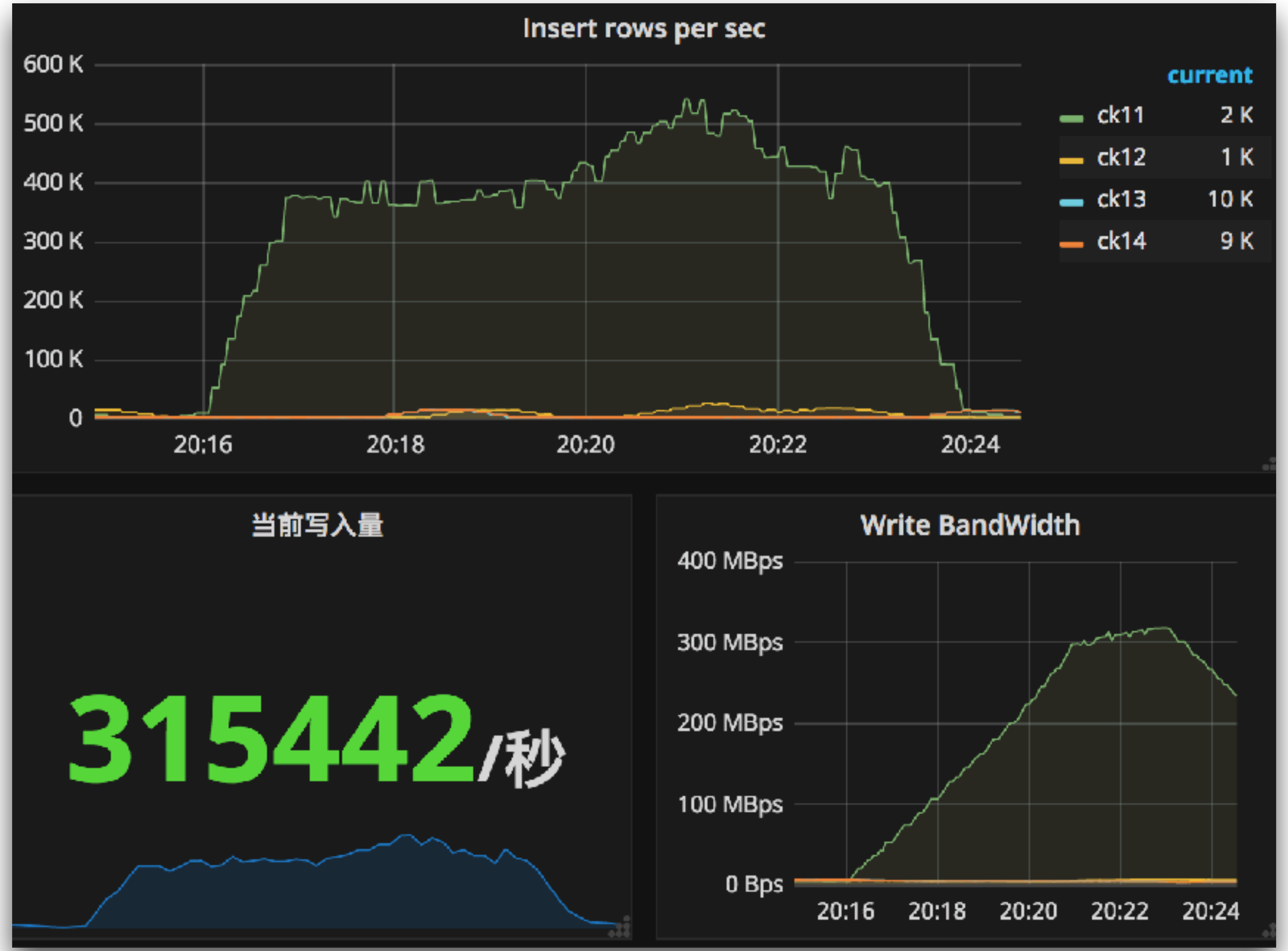


部署： 单机

响应时间

```
SELECT elapsed
FROM processes
ORDER BY elapsed DESC
LIMIT 10

┌──────elapsed─┐
| 5.291229166 |
| 5.287489023 |
| 5.285718683 |
| 5.284233673 |
| 5.282636967 |
| 0.000210788 |
└─────────────┘
```

峰值50W QPS



部署： 单机

查询类型

1. 查询总量

2. 简单group by

```
:) select count(*)/100000000 from ontime ;

SELECT count(*) / 100000000
FROM ontime

┌─divide(count(), 100000000)─┐
│                 1.71412868 │
└────────────────────────────┘

1 rows in set. Elapsed: 0.051 sec. Processed 171.41 million rows, 171.41 MB (3.35 billion rows/s., 3.35 GB/s.)

:) select Year, count(*) as c1 from ontime group by Year limit 3
:-] ;

SELECT
    Year,
    count(*) AS c1
FROM ontime
GROUP BY Year
LIMIT 3

┌─Year─┬──────c1─┐
│ 1988 │ 5202084 │
│ 1989 │ 5041188 │
│ 1990 │ 5270881 │
└──────┴─────────┘

3 rows in set. Elapsed: 0.208 sec. Processed 171.41 million rows, 342.83 MB (825.43 million rows/s., 1.65 GB/s.)
```

https://clickhouse.yandex/docs/en/getting_started/example_datasets/ontime.html

部署：单机

查询类型

- 条件查询，聚合，排序

```
SELECT
    DestCityName,
    uniqExact(OriginCityName) AS u
FROM ontime
WHERE (Year >= 2000) AND (Year <= 2010)
GROUP BY DestCityName
ORDER BY u DESC
LIMIT 10

┌─DestCityName──────────┬───u─┐
│ Atlanta, GA           │ 193 │
│ Chicago, IL           │ 167 │
│ Dallas/Fort Worth, TX │ 161 │
│ Minneapolis, MN       │ 138 │
│ Cincinnati, OH        │ 138 │
│ Detroit, MI           │ 130 │
│ Houston, TX           │ 129 │
│ Denver, CO            │ 127 │
│ Salt Lake City, UT    │ 119 │
│ New York, NY          │ 115 │
└───────────────────────┴─────┘

10 rows in set. Elapsed: 1.185 sec. Processed 72.79 million rows, 3.37 GB (61.45 million rows/s., 2.85 GB/s.)
```

https://clickhouse.yandex/docs/en/getting_started/example_datasets/ontime.html

部署： 单机

查询类型

- 复杂查询

```sql
SELECT
    min(Year),
    max(Year),
    Carrier,
    count(*) AS cnt,
    sum(ArrDelayMinutes > 30) AS flights_delayed,
    round(sum(ArrDelayMinutes > 30) / count(*), 2) AS rate
FROM ontime
WHERE (DayOfWeek NOT IN (6, 7)) AND (OriginState NOT IN ('AK', 'HI', 'PR', 'VI')) AND (DestState NOT IN ('AK', 'HI', 'PR', 'VI')) AND (FlightDate < '2010-01-01')
GROUP BY Carrier
HAVING (cnt > 100000) AND (max(Year) > 1990)
ORDER BY rate DESC
LIMIT 1000
```

min(Year)	max(Year)	Carrier	cnt	flights_delayed	rate
2003	2009	EV	1454777	237698	0.16
2003	2009	B6	683874	103677	0.15
2006	2009	YV	740606	110389	0.15
2003	2009	FL	1082489	158748	0.15
2006	2009	XE	1016010	152431	0.15
2003	2005	DH	501056	69833	0.14
2001	2009	MQ	3238137	448037	0.14
2004	2009	OH	1195868	160071	0.13
2003	2006	RU	1007247	126733	0.13
1988	2009	UA	9593281	1197052	0.12
2003	2006	TZ	136735	16496	0.12
1988	2009	AA	10600421	1185336	0.11
1988	2009	CO	6029147	673863	0.11
1988	2001	TW	2659963	280741	0.11
1988	2009	DL	11869418	1156256	0.1
2003	2009	OO	2654259	257069	0.1
2007	2009	9E	577223	59437	0.1
1988	2009	NW	7601726	725460	0.1
1988	2009	US	10276931	991016	0.1
1988	2009	AS	1506003	146920	0.1
1988	2009	WN	12722172	1107840	0.09
1988	1991	PA	206841	19465	0.09
1988	2005	HP	2607603	235675	0.09
2005	2009	F9	307569	28679	0.09

24 rows in set. Elapsed: 1.094 sec. Processed 128.68 million rows, 1.57 GB (117.57 million rows/s., 1.44 GB/s.)

https://clickhouse.yandex/docs/en/getting_started/example_datasets/ontime.html

部署： 单机

使用

1. 启动Server
2. use db, create table
3. 尽情select
4. 推荐引擎：MergeTree

```
CREATE TABLE apm.apm_msg (_clientip String,  _data_size Float32,
    date Date,  ts DateTime, hour Int8, minute Int8)
ENGINE = MergeTree(date, (minute, hour, date), 8192);
```

分区　　主键　　稀疏索引粒度

总结

优点：

1. 部署简单
2. 全部CPU打满，查询效率极高

问题:

1. 性能依赖单机（scale up路线）
2. 存在单点故障风险（宕机数据全丢）

MergeTree

写

- 如何写的快?
- 是否可压缩?

类似LSM Tree，但是没有内存表，不记录log

直接落磁盘，按照主键排序，分块写入

异步merge，与写不冲突，最大merge到月纬度

不支持删除、修改

primary.idx+*.bin+*.mrk+checksums.txt+columns.txt

读

- 如何快速查找?
- 数据量大，如何适应内存?

- 主键查询:

eg: (x, y, z, date)

每8192行，抽取一行数据
形成稀疏索引
最左原则？后面讨论

- 非主键查询:

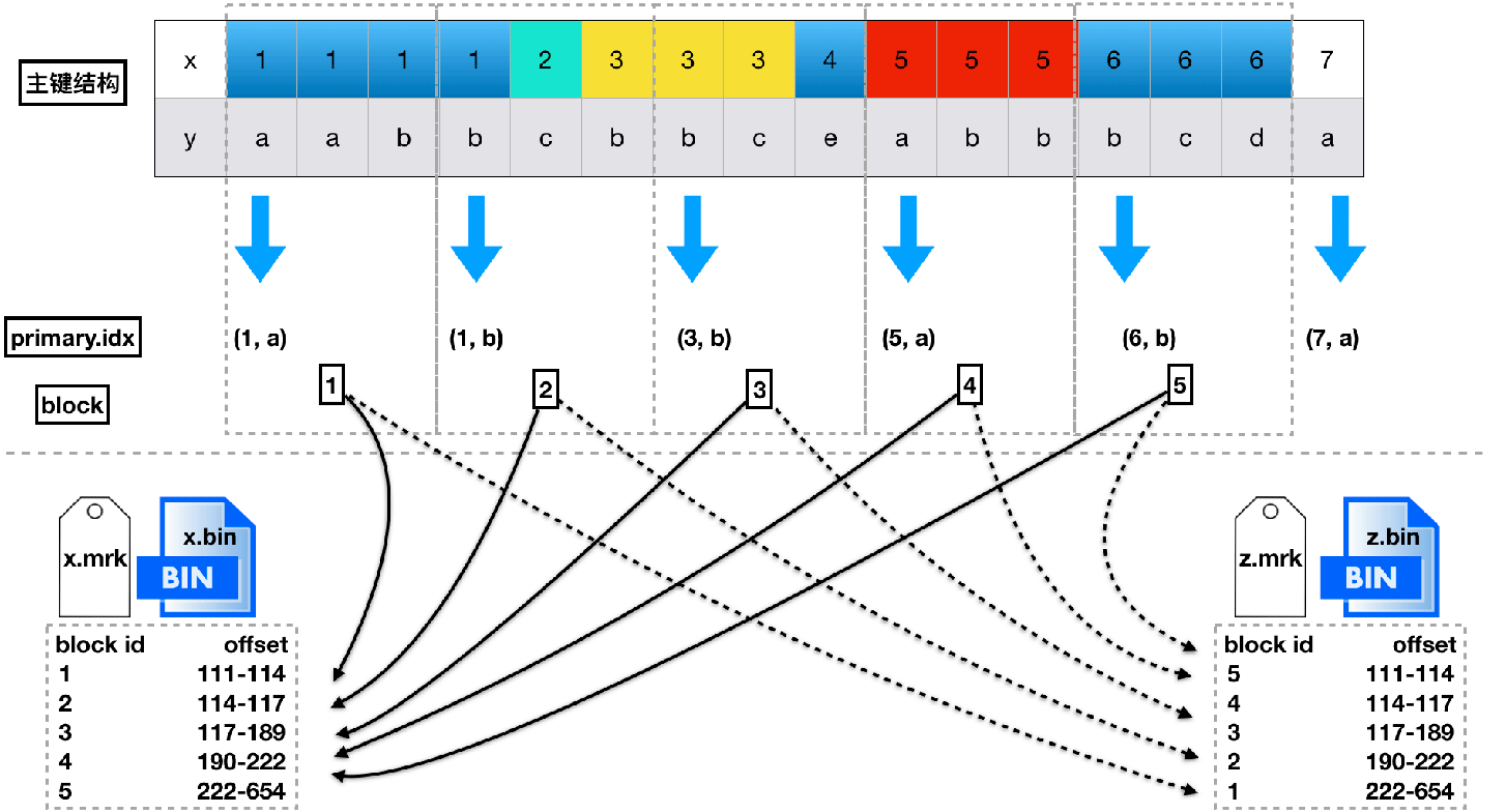
向量化操作
性能奇高

列式存储+向量化=超高性能

MergeTree

存储结构

```
├── data
|   ├── default
|   ├── gaopeng4
|   |   └── apm_msg
|   |       ├── 20171001_20171001_0
|   |       |   ├── checksums.txt
|   |       |   ├── columns.txt
|   |       |   ├── date.bin
|   |       |   ├── date.mrk
|   |       |   ├── minute.bin
|   |       |   ├── minute.mrk
|   |       |   └── primary.idx
|   |       └── detached
|   └── system
```

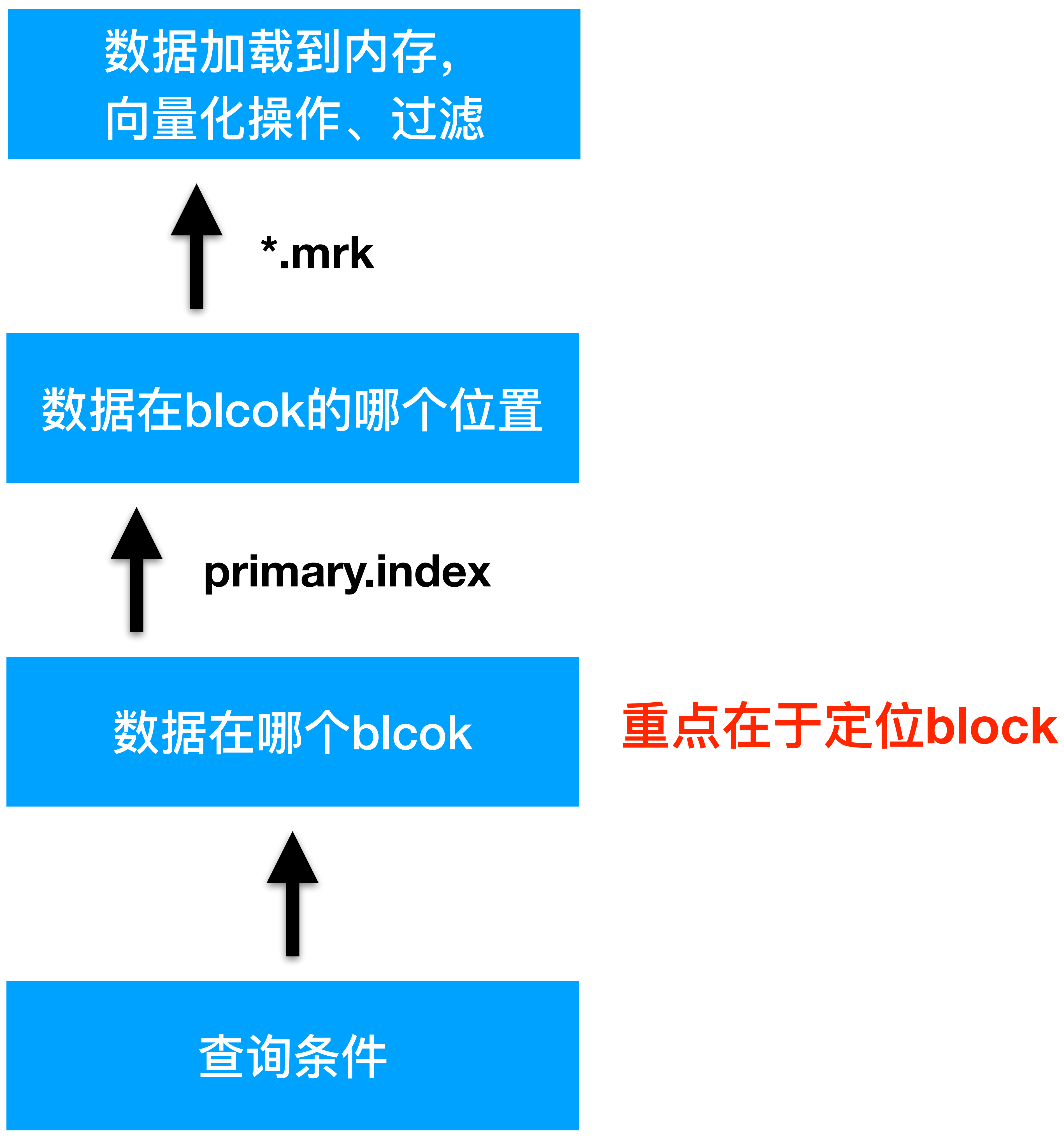


MergeTree

查询逻辑

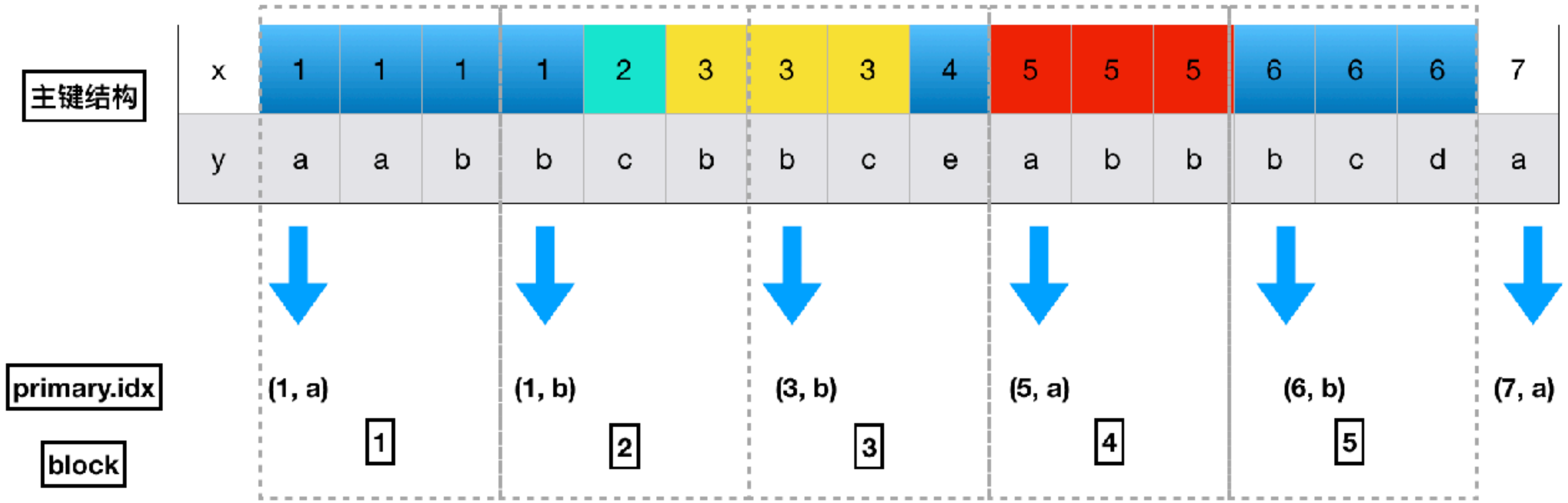
1. 并非使用BTree点对点查找，
稀疏索引，肯定存在大量无用数据过滤

2. 向量化操作



MergeTree

索引分析



全主键

where x='3' and y='c'

1. 判断，只需扫描block 2、3（定位block）
2. 使用mrk文件，定位到数据（找到数据）
3.加载内存过滤
4. 返回
5. y的作用呢？如果是(1, c) (1, c) (1, d) (1, e)呢?

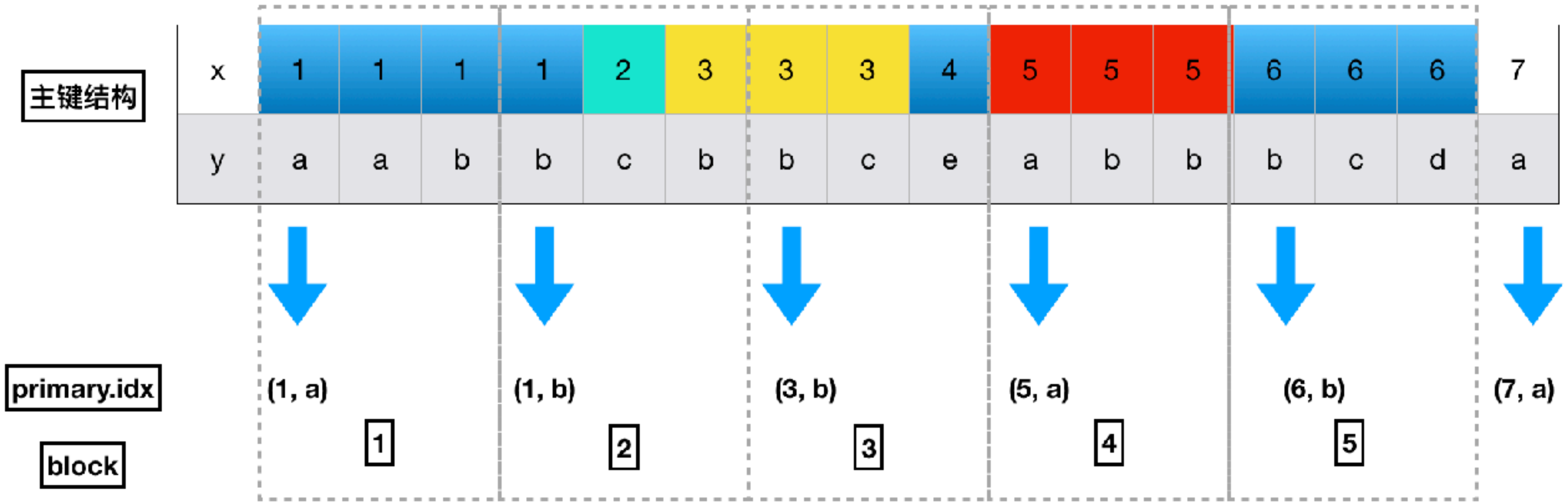
半主键

where x='3'
只扫描block 2、3

where y='c'
1. block 1首先被过滤掉（(1,a) (1,b) 1=1）
2. 所需block 2、3、4、5 （定位block）
3. 剩余过程类似
4. 该情况下，存在过滤效果差的情况

MergeTree

索引分析



非主键

where z='?'

如何定位z的数据?

等效于

where x=any and y=any and z='?'

1. 所有block（定位block）
2. 取所有mrk里所有的数据偏移指向，即全扫描
3. 过滤

主键+非主键

where x='?' and z='?'

1. 利用主键x，找到x的block，同时也一定是z要过滤的block（定位block）
2. 取出x、z mrk文件里的偏移量（定位数据）
3. 加载、过滤
4. 返回

MergeTree

索引分析

1. 如何定位Block

2. 向量化过滤

全主键

where x='3' and y='c'

1. 判断，只需扫描block 2、3（定位block）
2. 使用mrk文件，定位到数据（找到数据）
3. 加载内存过滤
4. 返回
5. y的作用呢？如果是(1, c) (1, c) (1, d) (1, e)呢？

半主键

where x='3'
只扫描block 2、3

where y='c'
1. block 1首先被过滤掉（(1,a) (1,b) 1=1）
2. 所需block 2、3、4、5 （定位block）
3. 剩余过程类似
4. 该情况下，存在过滤效果差的情况

非主键

where z='?'
如何定位z的数据？
等效于
where x=any and y=any and z='?'
1. 所有block（定位block）
2. 取所有mrk里所有的数据偏移指向，即全扫描
3. 过滤

主键+非主键

where x='?' and z='?'

1. 利用主键x，找到x的block，同时也一定是z要过滤的block（定位block）
2. 取出x、z mrk文件里的偏移量（定位数据）
3. 加载、过滤
4. 返回

MergeTree

索引建议

1. 主键设计很关键，但是不用像MySQL那么操心

2. 实际使用，“差不多就行了”

3. 实际使用，务必增加date=xxx字段，
通过分区过滤，即使用了很多非主键，
实际查询效果对OLAP来说，完全可以接受

Query: where date=? and hour=? and xxx=?

Index A (date, hour, min, ts)

Index B (ts, min, hour, date)

A is Good.

缺乏：

扩展性

可靠性

如何获得：

扩展性

可靠性

部署：‘分布式’

概括

假的‘scale out’

借助于特殊引擎实现

借助配置文件

部署：‘分布式’

Distributed引擎：

1. 本身不存储数据

2. 被写入，做转发

3. 查询，作为中间件，聚合后返回给用户

```
CREATE TABLE apm.apm_msg (_clientip String,  _data_size Float32,
    date Date,  ts DateTime, hour Int8, minute Int8)
ENGINE = MergeTree(date, (minute, hour, date), 8192);
```

分区　主键　颗粒度

```
CREATE TABLE apm.apm_msg_all
(_clientip String,  _data_size Float32,
    date Date,  ts DateTime, hour Int8, minute Int8)
ENGINE = Distributed(bip_ck_cluster, apm, apm_msg, rand());
```

集群名称　库　表　分布算法

部署：‘分布式’

- 分布式如何做到的

```xml
<clickhouse_remote_servers>
    <bip_ck_cluster>
        <shard>
            <internal_replication>true</internal_replication>
            <replica>
                <host>ck11.[redacted].com.cn</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <replica>
                <internal_replication>true</internal_replication>
                <host>ck12.[redacted].com.cn</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <internal_replication>true</internal_replication>
            <replica>
                <host>ck13.[redacted].com.cn</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <internal_replication>true</internal_replication>
            <replica>
                <host>ck14.[redacted].com.cn</host>
                <port>9000</port>
            </replica>
        </shard>
    </bip_ck_cluster>
</clickhouse_remote_servers>
```

分片1

分片2

分片3

分片4

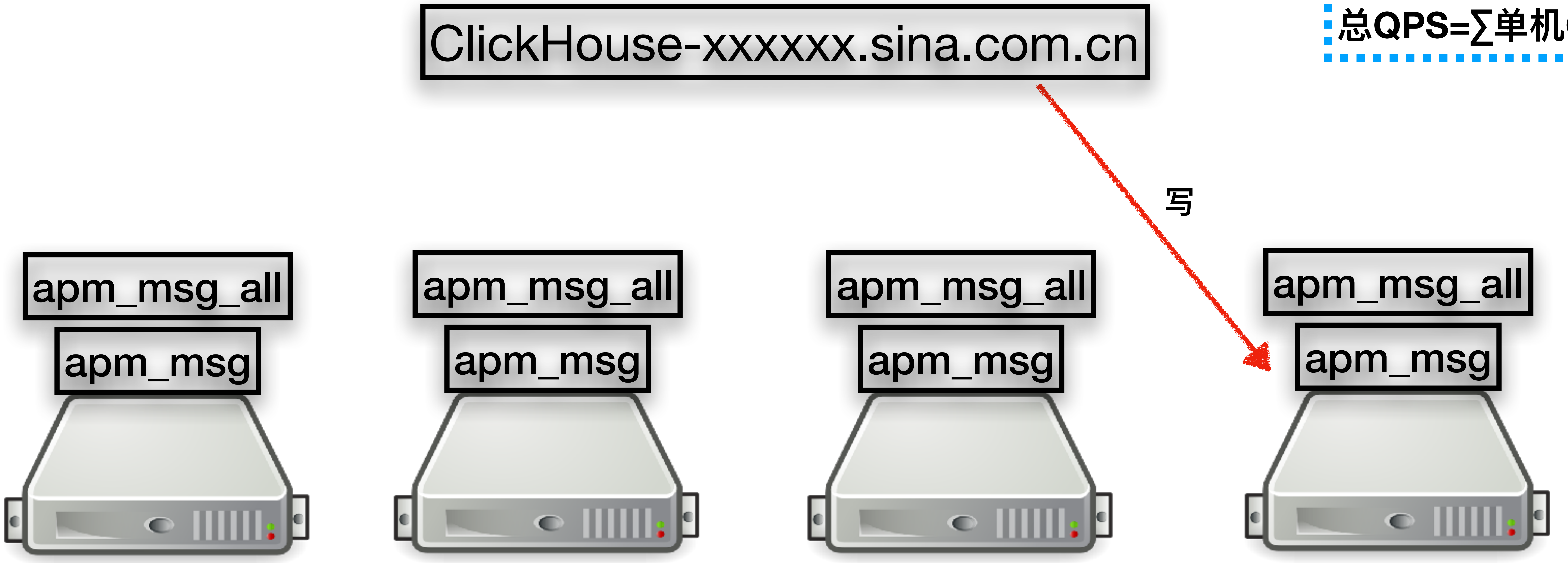
部署：‘分布式’

写操作

通过域名，写本地表

insert into apm_msg values()

总QPS=∑单机QPS

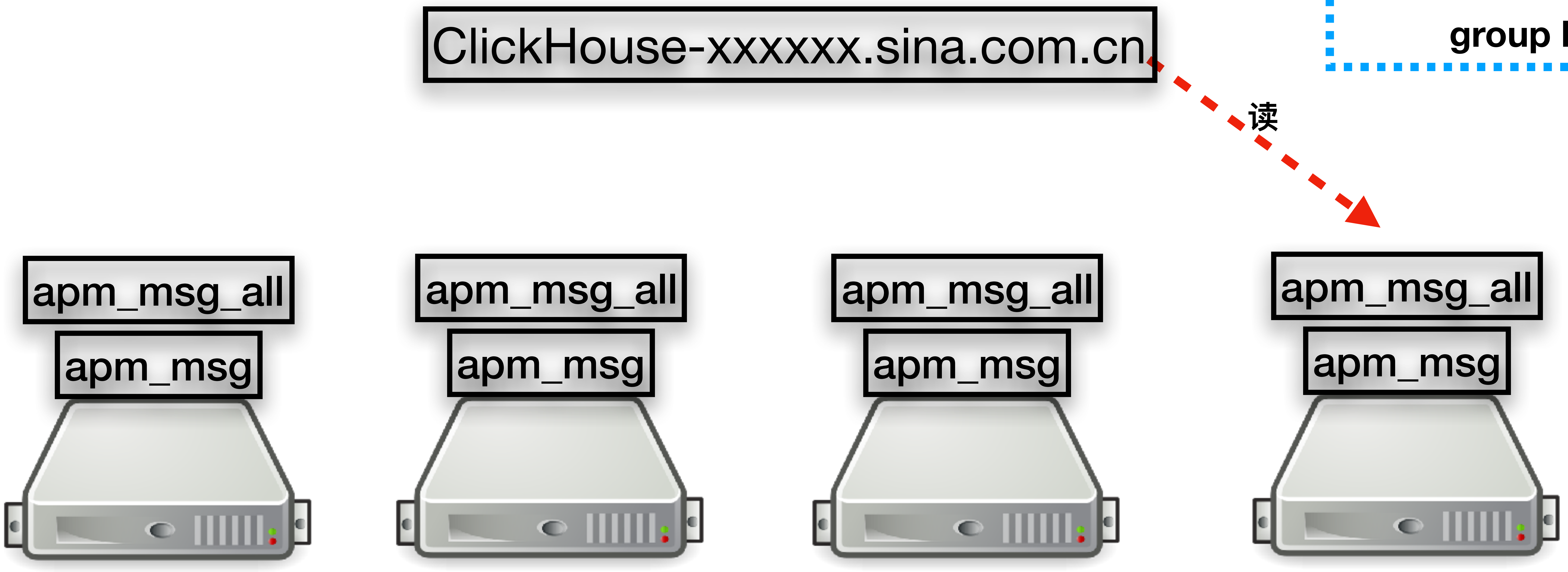


部署：‘分布式’

读操作

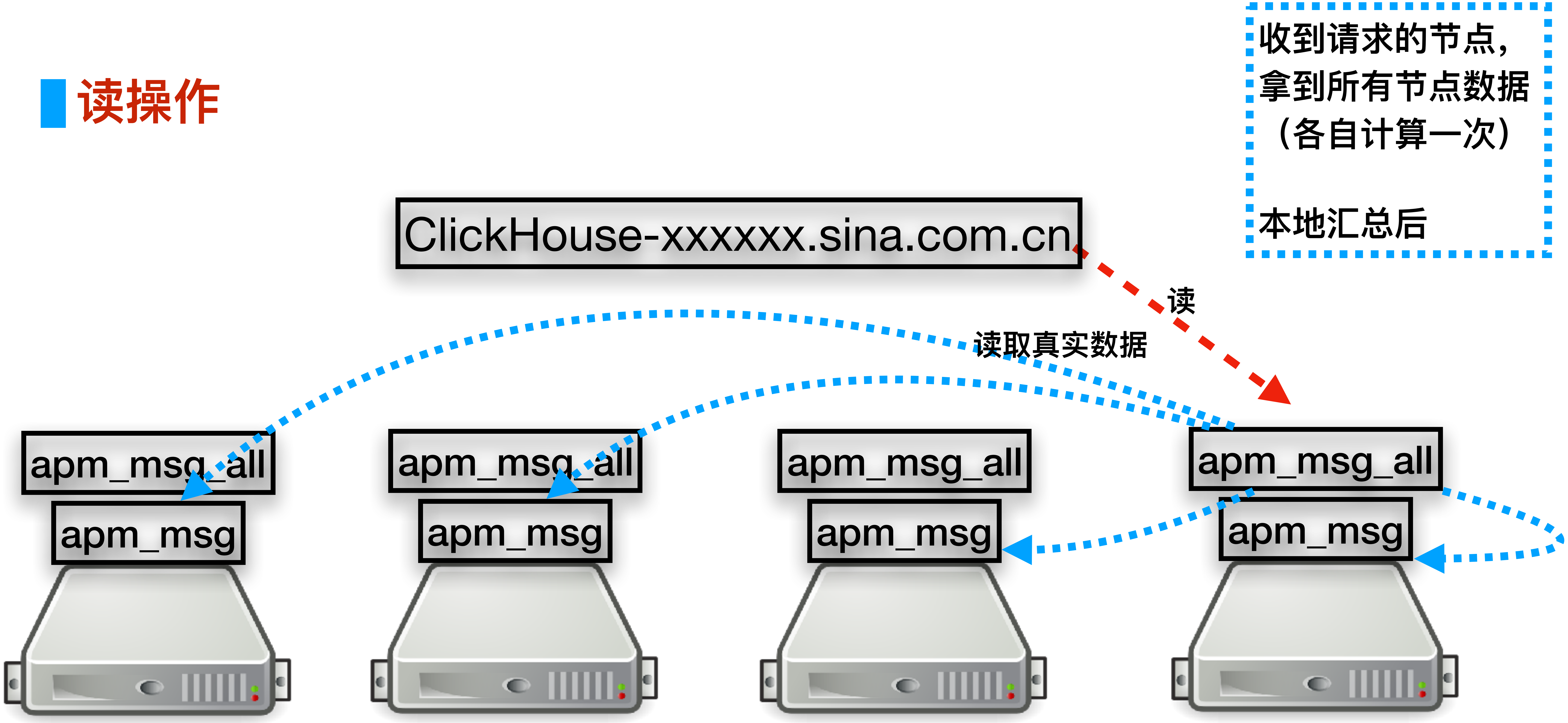
通过域名，读分布式表
select * from apm_msg_all
where xxx=yyy

原则：确保其他节点返回的数据，自己还可以聚合，如top/group by逻辑就不同



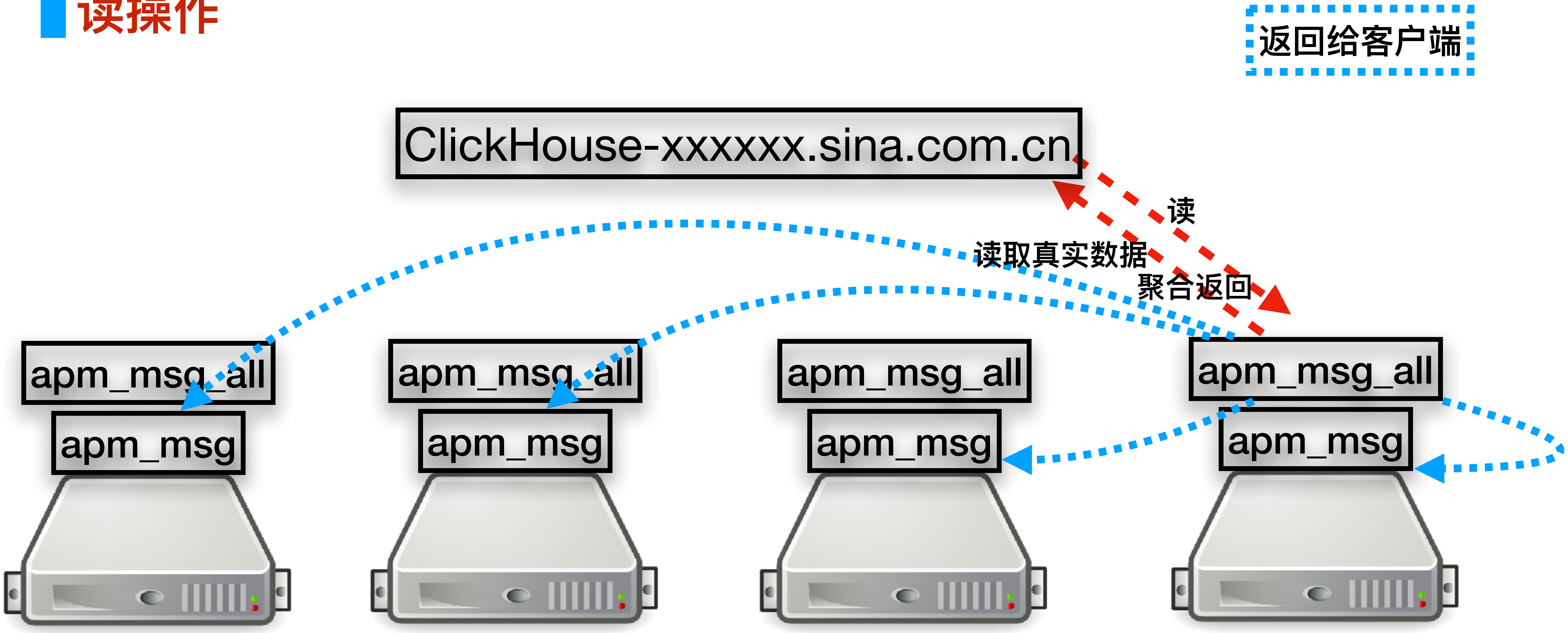
部署：‘分布式’

读操作


收到请求的节点，
拿到所有节点数据
（各自计算一次）
本地汇总后
ClickHouse-xxxxxx.sina.com.cn
读
读取真实数据
apm_msg_all
apm_msg
apm_msg_all
apm_msg
apm_msg_all
apm_msg
apm_msg_all
apm_msg

部署：‘分布式’

读操作



返回给客户端
ClickHouse-xxxxxx.sina.com.cn
读
读取真实数据
聚合返回
apm_msg_all
apm_msg
apm_msg_all
apm_msg
apm_msg_all
apm_msg
apm_msg_all
apm_msg

部署：‘分布式’

总结

通过全局配置文件，达到集群相互知晓

各自维护各自的数据，让用户自己写入

水平扩展性很好

查询/写入能力随机器数线性增加

cluster config updated on the fly

部署：‘分布式’

问题

1. 直接写分布式表，造成数据不均匀

2. 域名映射的IP只有在初始解析

3. 新增节点，历史数据不会搬迁，造成不均衡

4. 过度的group by，导致大量数据交换

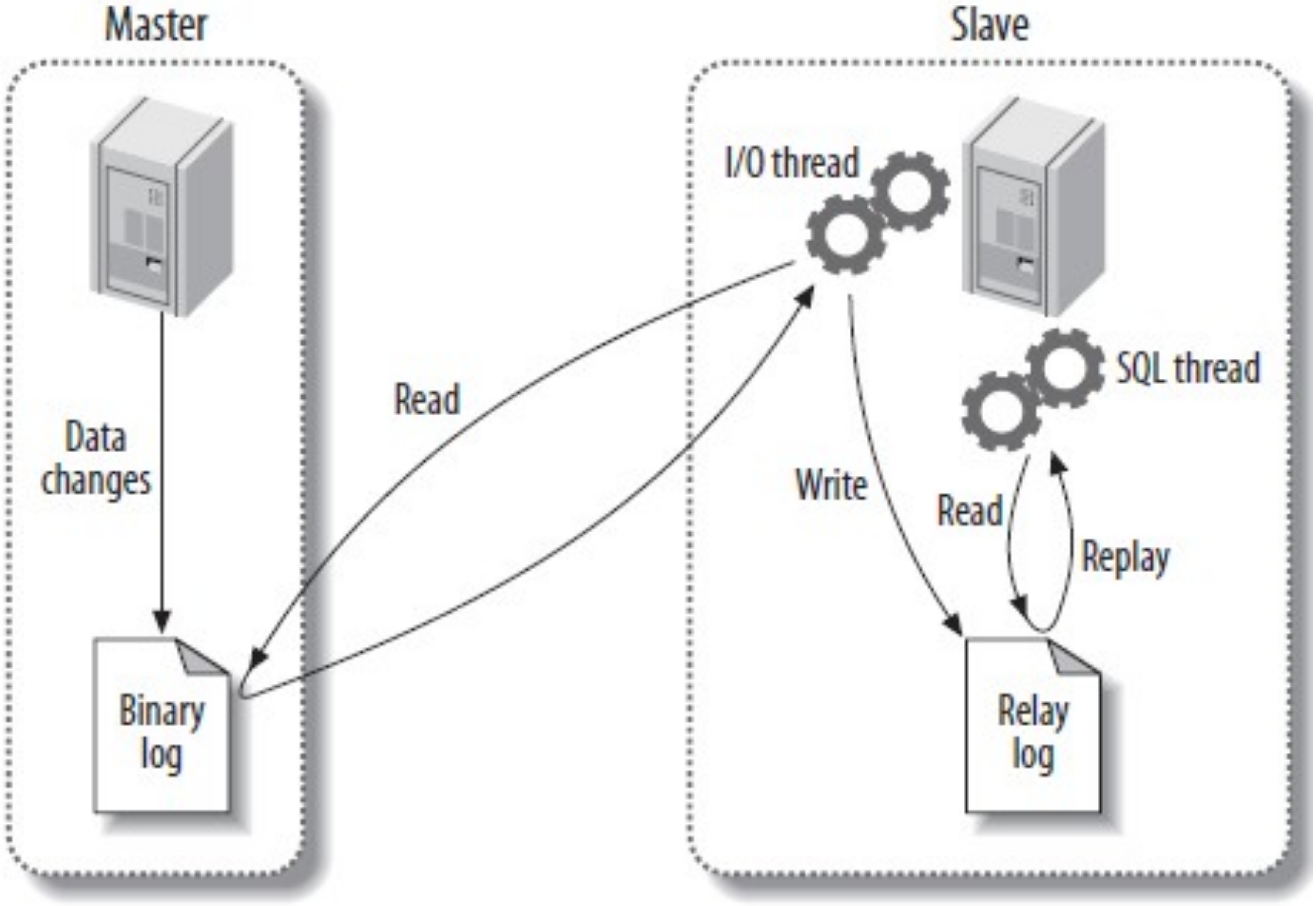
5. 数据分片依赖单机稳定性，缺乏可靠性

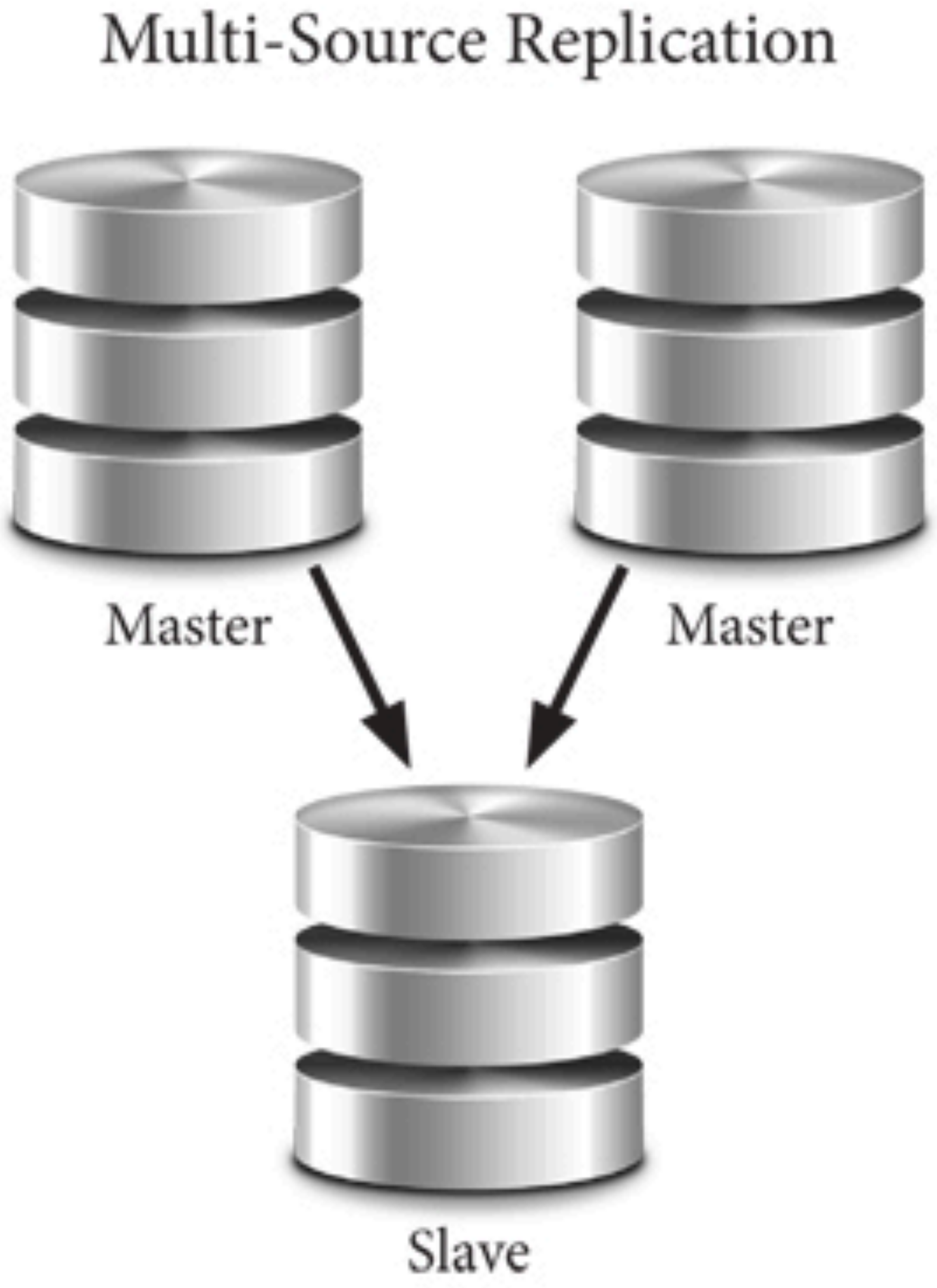
如何获得：

扩展性

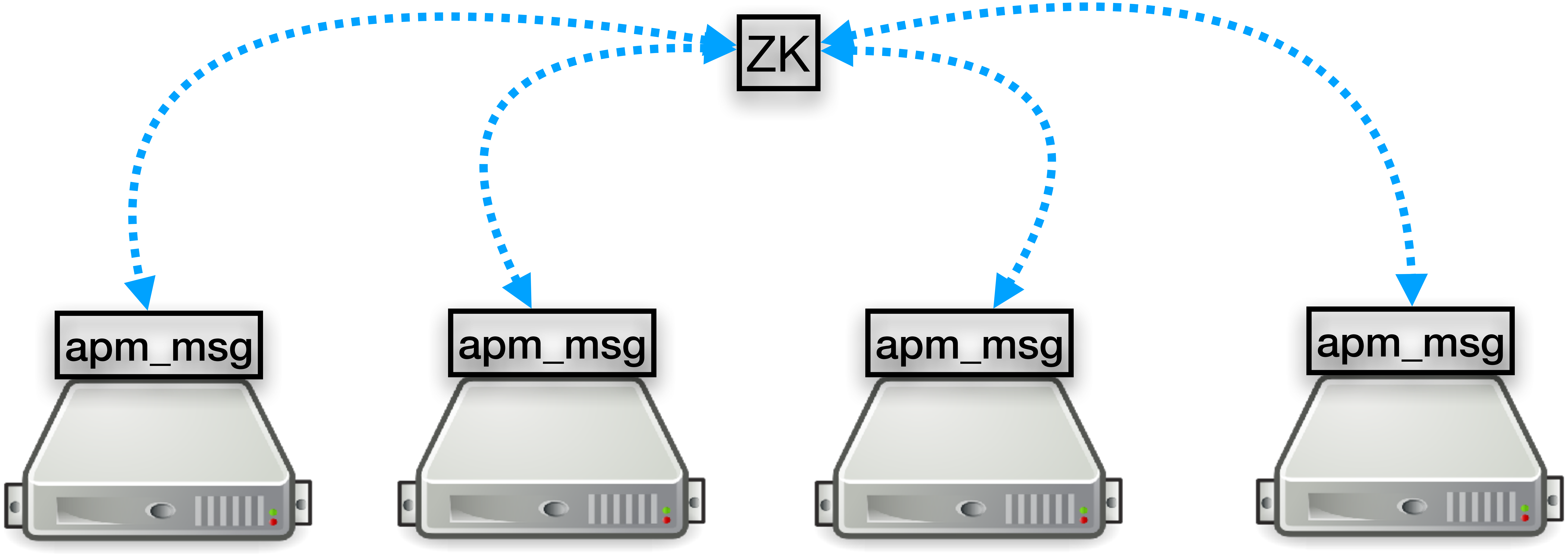
可靠性

我们知道的复制：


Master
Slave
I/O thread
Data
changes
Read
Write
SQL thread
Read
Replay
Binary
log
Relay
log


Multi-Source Replication
Master
Master
Slave

ClickHouse的复制



多源、多主、多向复制

数据‘互通有无’

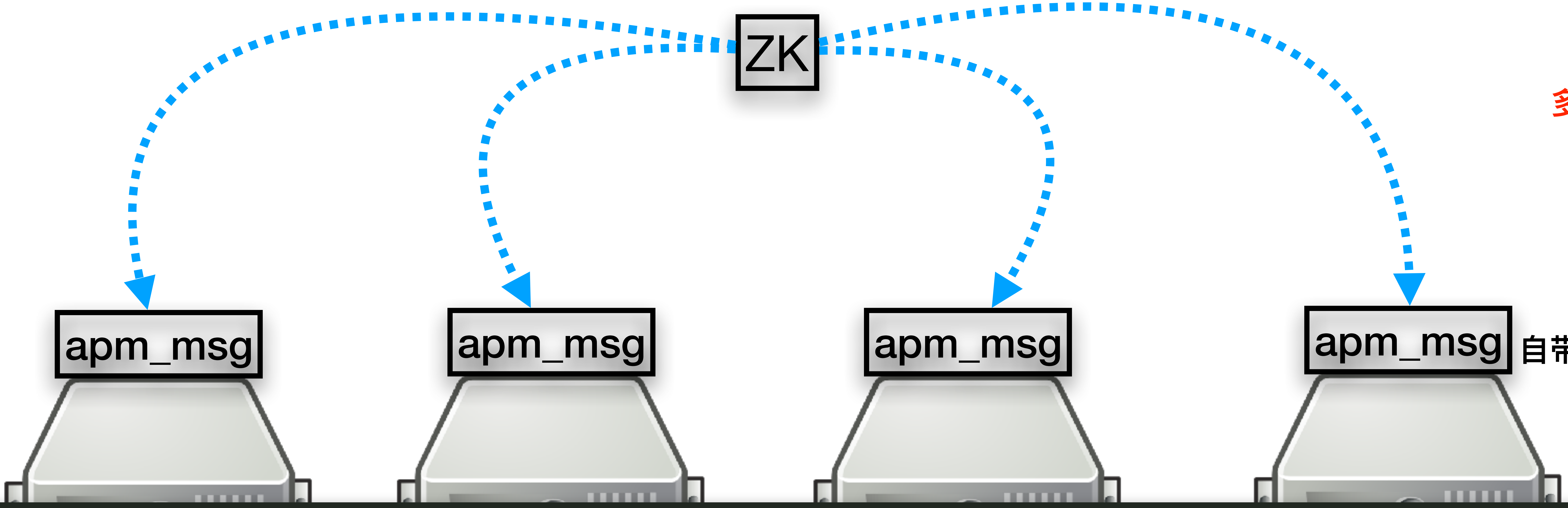
自带检测机制

自带同步机制（物理复制）

依赖ZK

非多数派写

部署： 复制



多源、多主、多向复制

数据‘互通有无’

自带检测机制

自带同步机制（物理复制）

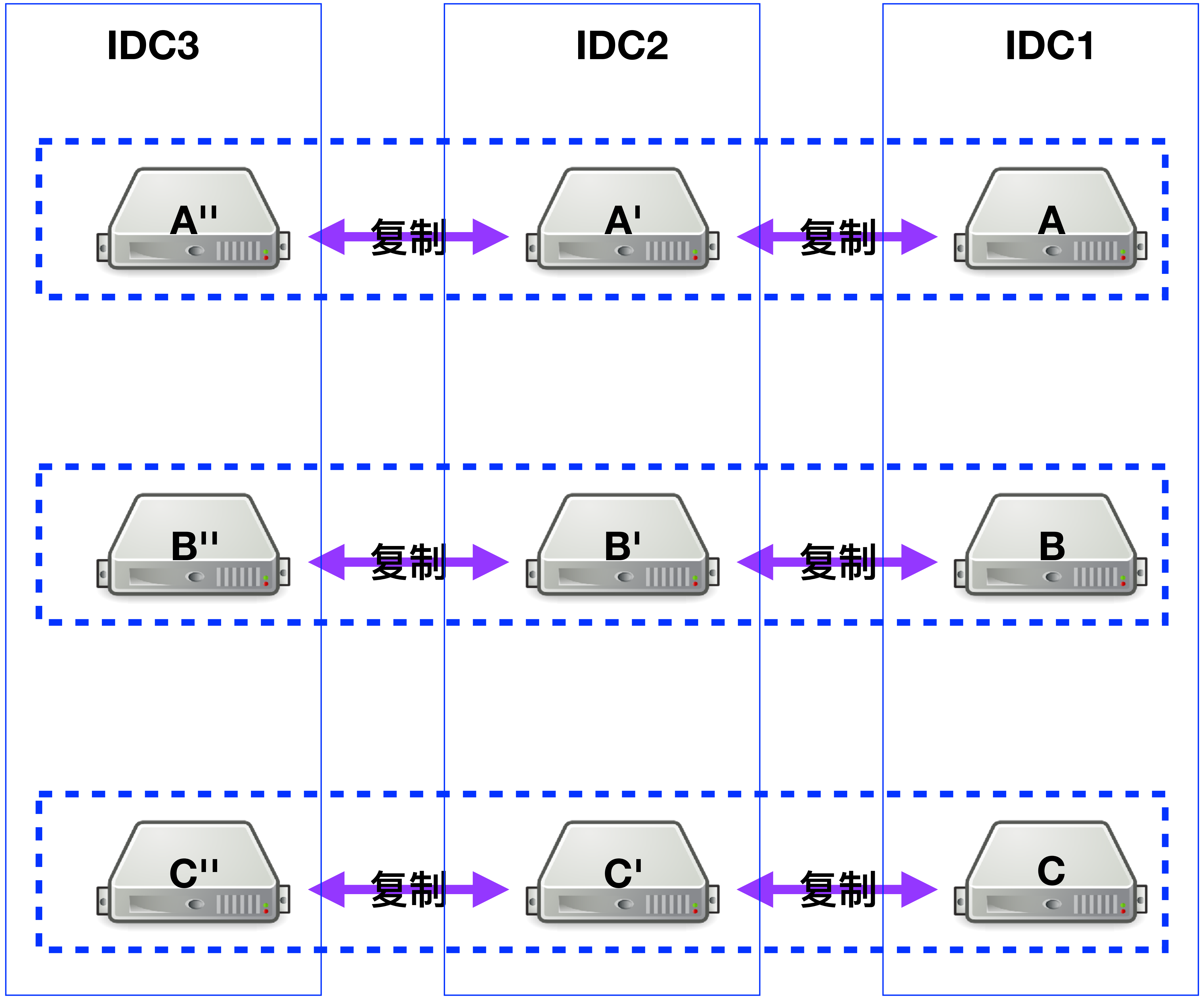
依赖ZK

非多数派写

```
CREATE TABLE apm.apm_msg (_clientip String,  _data_size Float32,
    date Date,  ts DateTime, hour Int8, minute Int8)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/apm_msg', '195', date, (minute, hour, date), 8192);
```

ReplicatedMergeTree('zk路径', '副本名称', 日期列, (其他列, 日期列), 索引粒度)

部署：最佳架构

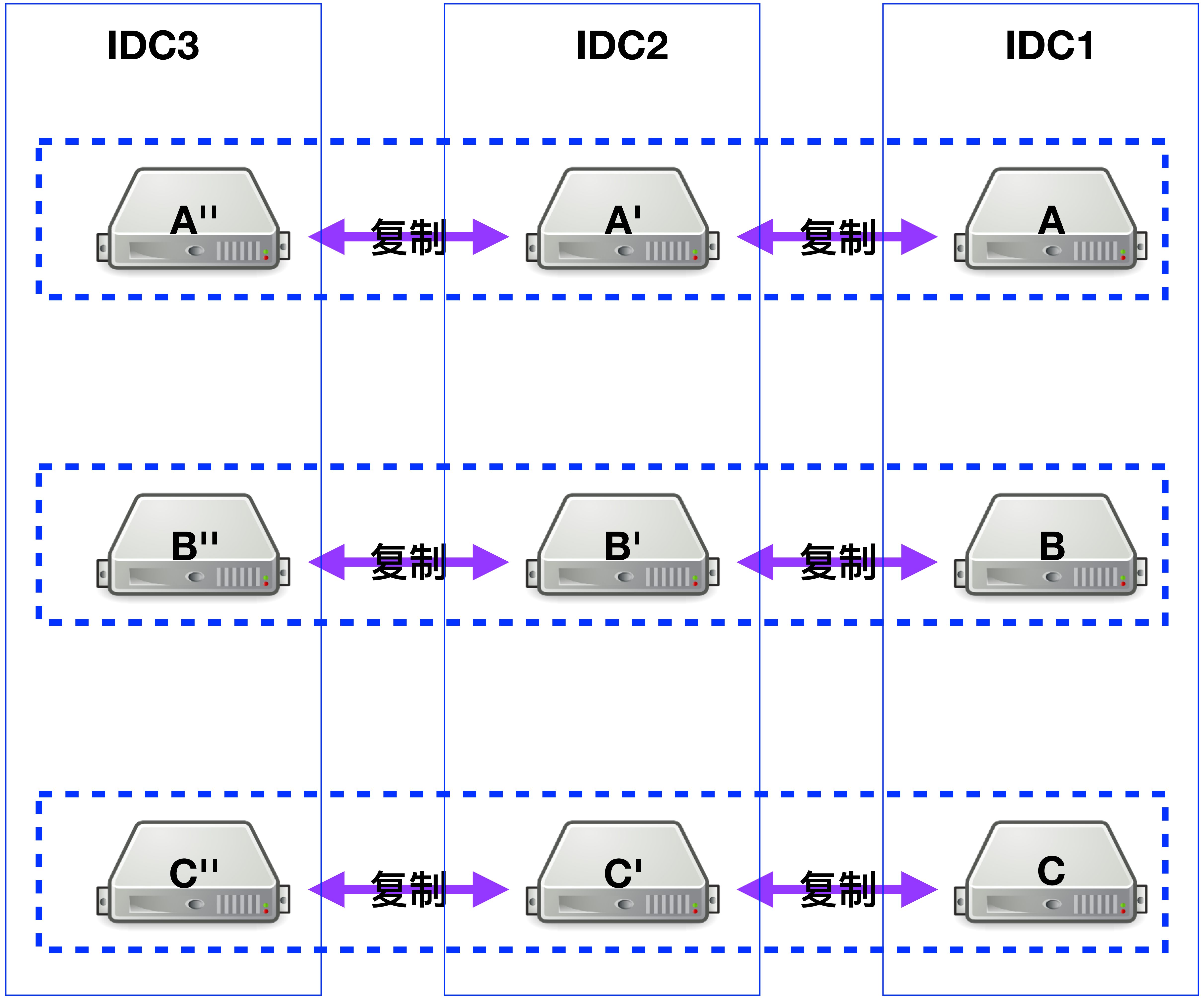


3个IDC使用复制机制做互备

每个IDC 3个节点，做分布式表，分担查询压力

如：
A/B/C 分别是数据的3个分片，各自承担1/3的数据
查询性能：A+B+C

部署：最佳架构



宕机分析

1. 3个IDC，挂掉2个
不影响读写
恢复只需重启实例
ClickHouse自动完成数据同步

2. 挂掉某个实例，如A
切换读写到其他IDC，恢复实例A即可

3. 挂掉多个实例，A、B同时挂掉
处理方式同上

集群的ClickHouse有多快?

我们的架构

CPU E5-2620 @ 2.00GHz
开启超线程后24core

48G内存

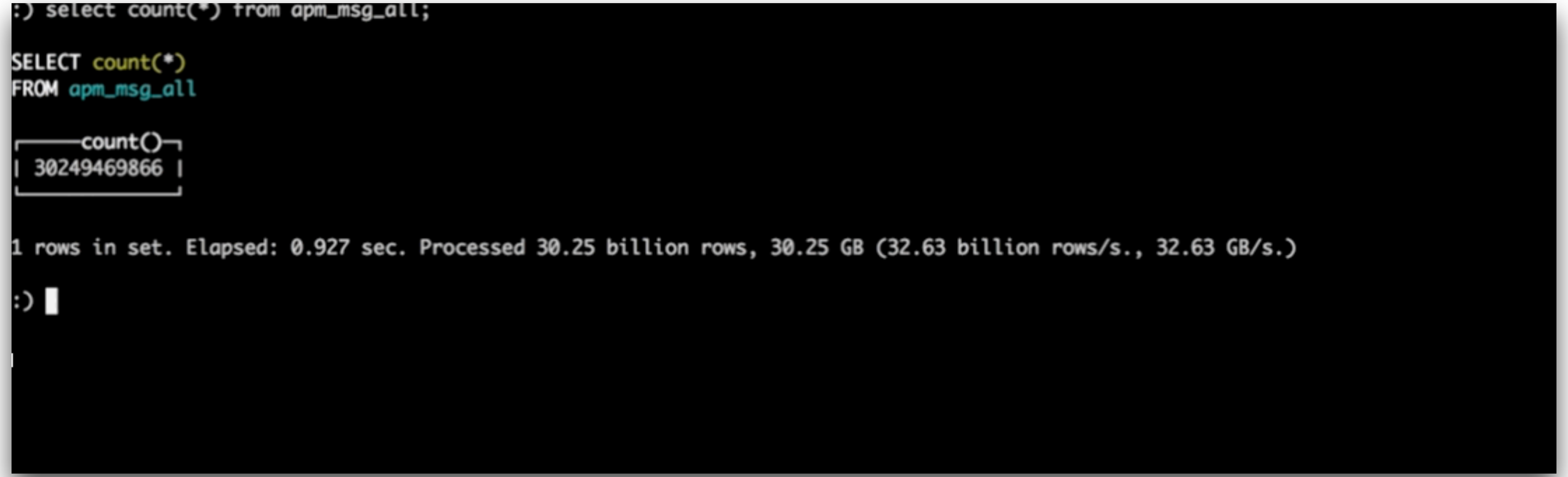
3T*12 Raid5

X 4

表1：
14字段 1200亿 4T

表2：
94字段 700亿 15T

请开始你的表演：



select count(*)

300亿

0.9秒

请开始你的表演：

```
:) select date, count(*)/100000000 from apm_msg_all group by date order by date desc limit 10 ;

SELECT
    date,
    count(*) / 100000000
FROM apm_msg_all
GROUP BY date
ORDER BY date DESC
LIMIT 10

┌───────date─┬─divide(count(), 100000000)─┐
│ 2017-10-09 │                  2.95357962 │
│ 2017-10-08 │                  5.25704122 │
│ 2017-10-07 │                  5.02549947 │
│ 2017-10-06 │                  4.92416848 │
│ 2017-10-05 │                  4.98019803 │
│ 2017-10-04 │                  4.68522239 │
│ 2017-10-03 │                  4.81681714 │
│ 2017-10-02 │                  4.95085341 │
│ 2017-10-01 │                  4.78589495 │
│ 2017-09-30 │                  4.90037846 │
└────────────┴─────────────────────────────┘

10 rows in set. Elapsed: 9.872 sec. Processed 30.27 billion rows, 60.53 GB (3.07 billion rows/s., 6.13 GB/s.)

:)
```

select date, count(*) from xx
group by date

300亿group by日期

9.8秒

请开始你的表演：

```
            _request_url,
            count(1) AS hit_2hour
        FROM apm.apm_msg_all
        WHERE (date = toString(today())) AND (hour = (toInt8(substring(toString(now()), 12, 2)) - 2)) AND (_ek = '_error') AND (url_ty
e = 'PIC') AND (_http_code IN ('403', '404'))
        GROUP BY _request_url
    )
    ALL INNER JOIN
    (
        SELECT
            _request_url,
            round(MAX(hit) / 60, 2) AS maxhit_per_s,
            sum(hit) AS hit_hour
        FROM
        (
            SELECT
                _request_url,
                minute,
                COUNT(1) AS hit
            FROM apm.apm_msg_all
            WHERE (date = toString(today())) AND (hour = (toInt8(substring(toString(now()), 12, 2)) - 1)) AND (_ek = '_error') AND (ur
l_type = 'PIC') AND (_http_code IN ('403', '404'))
            GROUP BY
                _request_url,
                minute
        ) AS a
        GROUP BY _request_url
    ) USING (_request_url)
    GROUP BY
        _request_url,
        maxhit_per_s,
        hit_2hour,
        hit_hour
USING (_request_url)

Progress: 3.42 million rows, 213.70 MB (28.50 million rows/s., 1.78 GB/s.)  21%

Progress: 79.78 million rows, 6.18 GB (97.48 million rows/s., 7.55 GB/s.)  92%
```

```
10 rows in set. Elapsed: 1.140 sec. Processed 114.07 million rows, 9.46 GB (100.06 million rows/s., 8.29 GB/s.)
```

太复杂了

请开始你的表演：

```
SELECT
    hour,
    one_hit,
    two_hit,
    one_error_hit,
    two_error_hit,
    round((one_error_hit / ((one_hit / 0.3) + one_error_hit)) * 100, 2) AS one_error_rate,
    round((two_error_hit / ((two_hit / 0.3) + two_error_hit)) * 100, 2) AS two_error_rate
FROM
(
    SELECT
        hour,
        sum(caseWithoutExpr((area IN ('北京', '上海')) OR (city_name IN ('深圳', '广州')), 1, 0)) AS one_hit,
        sum(caseWithoutExpr((area NOT IN ('北京', '上海')) AND (city_name NOT IN ('深圳', '广州')), 1, 0)) AS two_hit
    FROM apm.apm_msg_all
    WHERE (date = toString(yesterday())) AND (_ek = '_network') AND (_snet != 'wifi') AND (country = '中国')
    GROUP BY hour
)
ANY LEFT JOIN
(
    SELECT
        hour,
        sum(caseWithoutExpr((area IN ('北京', '上海')) OR (city_name IN ('深圳', '广州')), 1, 0)) AS one_error_hit,
        sum(caseWithoutExpr((area NOT IN ('北京', '上海')) AND (city_name NOT IN ('深圳', '广州')), 1, 0)) AS two_error_hit
    FROM apm.apm_msg_all
    WHERE (date = toString(yesterday())) AND (_ek = '_error') AND (_snet != 'wifi') AND (country = '中国')
    GROUP BY hour
) USING (hour)

Progress: 558.25 million rows, 42.98 GB (102.60 million rows/s., 7.90 GB/s.)                89%
```

```
24 rows in set. Elapsed: 14.543 sec. Processed 1.23 billion rows, 94.76 GB (84.60 million rows/s., 6.52 GB/s.)

:)
```

更复杂了

支持Join

请开始你的表演：

官方demo

```
:) SELECT
:-]     count() as visits,
:-]     sum(PageViews) as hits,
:-]     uniq(UserID) as users
:-] FROM visits_all
:-] WHERE StartDate > today() - 7;

SELECT
    count() AS visits,
    sum(PageViews) AS hits,
    uniq(UserID) AS users
FROM visits_all
WHERE StartDate > (today() - 7)

┌──────visits─┬────────hits─┬──────users─┐
│ 12106577368 │ 46414744272 │ 1080418023 │
└─────────────┴─────────────┴────────────┘

1 rows in set. Elapsed: 6.245 sec. Processed 13.23 billion rows, 185.18 GB (2.12 billion rows/s., 29.65 GB/s.)
```

```
SELECT
    uniq(UserID) AS users,
    if(IsMobile, if(IsTablet, 'tablet', 'phone'), 'desktop') AS device
FROM visits_all
WHERE StartDate > (today() - 7)
GROUP BY device

┌─────users─┬─device─┐
│ 465363568 │ phone  │
└───────────┴────────┘
┌─────users─┬─device──┐
│ 548496870 │ desktop │
└───────────┴─────────┘
┌────users─┬─device─┐
│ 70608335 │ tablet │
└──────────┴────────┘

3 rows in set. Elapsed: 7.650 sec. Processed 13.23 billion rows, 158.71 GB (1.73 billion rows/s., 20.75 GB/s.)

:)
```

请开始你的表演：

官方压测PK

Compare: ClickHouse Vertica Vertica (x3) Vertica (x6) InfiniDB MonetDB Infobright Hive MySQL MemSQL Greenplum(x2) Greenplum

Dataset size: 10 mln. 100 mln. 1 bn.

Run number: first (cold cache) second third

Relative query processing time (lower is better):

Full results:

Query	ClickHouse (1.1.53960)		InfiniDB (Enterprise 3.6.23)		Hive (0.11, ORC File)		Greenplum (4.3.9.1)	
SELECT count() FROM hits	(0.297 s.)	(0.011 s.)	x6.97 (2.070 s.)	x30.91 (0.340 s.)	x372.65 (110.676 s.)	x9557.27 (105.130 s.)	x13.84 (4.110 s.)	x392.73 (4.320 s.)
SELECT count() FROM hits WHERE AdvEngine	(0.173 s.)	(0.009 s.)	x4.39 (0.760 s.)	x30.00 (0.300 s.)	x319.05 (55.195 s.)	x3643.50 (36.435 s.)	x14.80 (2.560 s.)	x253.00 (2.530 s.)
SELECT sum(AdvEngineID), count(), avg(Resol	(0.274 s.)	(0.065 s.)	x5.29 (1.450 s.)	x18.92 (1.230 s.)	x145.95 (39.991 s.)	x540.66 (35.143 s.)	x18.18 (4.980 s.)	x79.08 (5.140 s.)
SELECT sum(UserID) FROM hits	(0.552 s.)	(0.046 s.)	—	—	x80.55 (44.465 s.)	x741.98 (34.131 s.)	x12.03 (6.640 s.)	x130.22 (5.990 s.)
SELECT uniq(UserID) FROM hits	(0.485 s.)	(0.097 s.)	x8.62 (4.180 s.)	x40.10 (3.890 s.)	x228.23 (110.690 s.)	x1092.30 (105.953 s.)	x12.08 (5.860 s.)	x58.97 (5.720 s.)
SELECT uniq(SearchPhrase) FROM hits	(0.625 s.)	(0.211 s.)	x42.11 (26.320 s.)	x133.03 (28.070 s.)	x108.99 (68.119 s.)	x307.26 (64.831 s.)	x10.74 (6.710 s.)	x32.04 (6.760 s.)
SELECT min(EventDate), max(EventDate) FROI	(0.164 s.)	(0.047 s.)	x8.29 (1.360 s.)	x22.13 (1.040 s.)	x230.54 (37.809 s.)	x702.57 (33.021 s.)	x25.91 (4.250 s.)	x93.40 (4.390 s.)
SELECT AdvEngineID, count() FROM hits WHE	(0.056 s.)	(0.009 s.)	x10.00 (0.560 s.)	x32.00 (0.320 s.)	x960.50 (53.788 s.)	x5126.10 (51.261 s.)	x80.89 (4.530 s.)	x423.00 (4.230 s.)
SELECT RegionID, uniq(UserID) AS u FROM hi	(0.966 s.)	(0.427 s.)	x5.32 (5.140 s.)	x10.63 (4.540 s.)	x90.56 (87.479 s.)	x199.21 (85.062 s.)	x9.44 (9.120 s.)	x21.62 (9.230 s.)
SELECT RegionID, sum(AdvEngineID), count()	(0.970 s.)	(0.508 s.)	x8.07 (7.830 s.)	x16.10 (8.180 s.)	x109.87 (106.577 s.)	x202.52 (102.879 s.)	x15.23 (14.770 s.)	x30.28 (15.380 s.)
SELECT MobilePhoneModel, uniq(UserID) AS u	(0.620 s.)	(0.168 s.)	x3.16 (1.960 s.)	x8.33 (1.400 s.)	x97.42 (60.400 s.)	x318.44 (53.498 s.)	x5.19 (3.220 s.)	x17.08 (2.870 s.)
SELECT MobilePhone, MobilePhoneModel, unic	(0.655 s.)	(0.187 s.)	x2.67 (1.750 s.)	x8.13 (1.520 s.)	x93.55 (61.275 s.)	x287.16 (53.698 s.)	x6.73 (4.410 s.)	x22.94 (4.290 s.)

快

快

快

为啥这么快?

1. 优秀的代码编写，强大的底层优化，严格的单元测试

2. A vector engine （多）& Code generation（少）

3. CPU底层指令集（SIMD）的使用

4. 列式存储、牺牲事务、MPP架构

为啥这么快?

参照阅读

向量化与 计算

PgSQL · 引擎介绍 · 向量化执行引擎简介

Vectorized processing

Data is represented as small single-dimensional arrays (vectors), easily accessible for CPUs.

The percentage of instructions spent in interpretation logic is reduced by a factor equal to the vector-size

The functions that perform work now typically process an array of values in a tight loop

Tight loops can be optimized well by compilers, enable compilers to generate SIMD instructions automatically.

Modern CPUs also do well on such loops, out-of-order execution in CPUs often takes multiple loop iterations into execution concurrently, exploiting the deeply pipelined resources of modern CPUs.

It was shown that vectorized execution can improve data-intensive (OLAP) queries by a factor 50.

ClickHouse函数：

300+

ClickHouse高级函数：

统计类

quantile(0.99)(X)

quantiles(0.9, 0.99, 0.999)(X)

median(X)

varSamp(X)

stddevSamp(X)

URL截取

cutQueryString(X)

domain(X)

其他

today()-1

yesterday()

substring(s, offset, length)

IPv4NumToStringClassC

extract(haystack, pattern)

高级函数举例：

- 域名类

```
SELECT protocol('https://weibo.com/jackpgao/home?wvr=5') as protocol;
┌─protocol─┐
│ https    │
└──────────┘

SELECT domain('https://weibo.com/jackpgao/home?wvr=5') as domain;
┌─domain────┐
│ weibo.com │
└───────────┘

SELECT domainWithoutWWW('https://weibo.com/jackpgao/home?wvr=5') as domainWithoutWWW;
┌─domainWithoutWWW─┐
│ weibo.com        │
└──────────────────┘

SELECT cutQueryString('https://weibo.com/jackpgao/home?wvr=5') as cutQueryString;
┌─cutQueryString──────────────────┐
│ https://weibo.com/jackpgao/home │
└─────────────────────────────────┘

SELECT path('https://weibo.com/jackpgao/home?wvr=5') as path;
┌─path────────────┐
│ /jackpgao/home  │
└─────────────────┘

SELECT pathFull('https://weibo.com/jackpgao/home?wvr=5') as pathFull;
┌─pathFull───────────────┐
│ /jackpgao/home?wvr=5   │
└────────────────────────┘
```

高级函数举例：

- IP归类

```
SELECT
    IPv4NumToStringClassC(IPv4StringToNum(clientip)) AS k,
    count()
FROM xxx
WHERE (date = toString(today())) AND (hour = 12)
GROUP BY k
ORDER BY count() DESC
LIMIT 10

┌─k──────────────┬─count()─┐
│ 118.190.218.xxx │   99353 │
│ 59.63.248.xxx   │   64207 │
│ 122.224.52.xxx  │   49156 │
│ 117.136.79.xxx  │   47350 │
│ 120.241.4.xxx   │   47041 │
│ 59.63.249.xxx   │   45717 │
│ 115.124.31.xxx  │   40806 │
│ 111.13.31.xxx   │   38786 │
│ 223.104.3.xxx   │   37271 │
│ 0.0.0.xxx       │   36596 │
└────────────────┴─────────┘
```

高级函数举例：

- 百分比响应时间

```
SELECT quantiles(0.9, 0.99, 0.999)(t_total)
FROM analyse_msg
WHERE rs LIKE '%mysql%' and  date=toString(today())

┌─quantiles(0.9, 0.99, 0.999)(t_total)─────────────────────┐
│ [1.5299999713897705,37.60140113830588,694.099821991015] │
└──────────────────────────────────────────────────────────┘

1 rows in set. Elapsed: 0.056 sec. Processed 236.86 thousand rows, 14.17 MB (4.25 million rows/s., 254.39 MB/s.)


SELECT quantiles(0.9, 0.99, 0.999)(t_total)
FROM analyse_msg
WHERE rs LIKE '%redis%' and  date=toString(today())

┌─quantiles(0.9, 0.99, 0.999)(t_total)───────────────────────┐
│ [0.3690000057220513,1.5700000524520874,5.636179870128636] │
└────────────────────────────────────────────────────────────┘

1 rows in set. Elapsed: 0.041 sec. Processed 236.86 thousand rows, 14.30 MB (5.76 million rows/s., 347.84 MB/s.)
```

来点干货

目录结构

```
├── config-preprocessed.xml
├── config.xml
├── users-preprocessed.xml
└── users.xml
├── metrika.xml

├── cores
├── flags
├── status

├── tmp

├── log
|   ├── error.log
|   ├── server.log
|   ├── stderr
|   └── stdout

├── metadata
|   ├── apm.sql
|   ├── default
|   ├── gaopeng4
|   |   └── apm_msg.sql
|   └── system

├── data
|   ├── default
|   ├── gaopeng4
|   |   └── apm_msg
|   |       ├── 20171001_20171001_0_9_2
|   |       |   ├── checksums.txt
|   |       |   ├── columns.txt
|   |       |   ├── date.bin
|   |       |   ├── date.mrk
|   |       |   ├── minute.bin
|   |       |   ├── minute.mrk
|   |       |   └── primary.idx
|   |       └── detached
|   └── system
```

配置文件

状态文件

日志

元数据

数据目录

配置文件：

- config.xml

```xml
<?xml version="1.0"?>
<yandex>
    <logger>
        <level>trace</level>
        <log>/data1/clickhouse/log/server.log</log>
        <errorlog>/data1/clickhouse/log/error.log</errorlog>
        <size>1000M</size>
        <count>10</count>
    </logger>
    <http_port>8123</http_port>
    <tcp_port>9000</tcp_port>
    <interserver_http_port>9009</interserver_http_port>
    <interserver_http_host>ck21.[redacted].com.cn</interserver_http_host>
    <listen_host>0.0.0.0</listen_host>
    <max_connections>4096</max_connections>
    <keep_alive_timeout>3</keep_alive_timeout>
    <max_concurrent_queries>100</max_concurrent_queries>
    <uncompressed_cache_size>8589934592</uncompressed_cache_size>
    <mark_cache_size>10737418240</mark_cache_size>
    <path>/data1/clickhouse/</path>
    <tmp_path>/data1/clickhouse/tmp/</tmp_path>
    <users_config>users.xml</users_config>
    <default_profile>default</default_profile>

    <log_queries>1</log_queries>
    <default_database>default</default_database>
    <remote_servers incl="clickhouse_remote_servers" />
    <zookeeper incl="zookeeper-servers" optional="true" />
    <macros incl="macros" optional="true" />
    <builtin_dictionaries_reload_interval>3600</builtin_dictionaries_reload_interval>
    <max_table_size_to_drop>0</max_table_size_to_drop>
    <include_from>/data1/clickhouse/metrika.xml</include_from>
</yandex>
```

日志

本节点信息

本机域名

本地配置

集群相关配置

配置文件：

metrika.xml

```xml
<yandex>
<clickhouse_remote_servers>
    <bip_ck_cluster>
        <shard>
            <internal_replication>true</internal_replication>
            <replica>
                <host>ck11[redacted].com.cn</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <replica>
                <internal_replication>true</internal_replication>
                <host>ck12[redacted].com.cn</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <internal_replication>true</internal_replication>
            <replica>
                <host>ck13[redacted].com.cn</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <internal_replication>true</internal_replication>
            <replica>
                <host>ck14[redacted].com.cn</host>
                <port>9000</port>
            </replica>
        </shard>
    </bip_ck_cluster>
</clickhouse_remote_servers>
```

配置文件：

metrika.xml

```xml
<zookeeper-servers>
  <node index="1">
    <host>1.[redacted].sina.com.cn</host>
    <port>2181</port>
  </node>
  <node index="2">
    <host>2.[redacted].sina.com.cn</host>
    <port>2181</port>
  </node>
  <node index="3">
    <host>3.[redacted].sina.com.cn</host>
    <port>2181</port>
  </node>
</zookeeper-servers>

<macros>
    <replica>ck11</replica>
</macros>

<networks>
   <ip>::/0</ip>
</networks>

<clickhouse_compression>
<case>
  <min_part_size>10000000000</min_part_size>
  <min_part_size_ratio>0.01</min_part_size_ratio>
  <method>lz4</method>
</case>
</clickhouse_compression>
</yandex>
```

配置文件：

- user.xml

```xml
<profiles>
    <default>
        <max_memory_usage>10000000000</max_memory_usage>
        <use_uncompressed_cache>0</use_uncompressed_cache>
        <load_balancing>random</load_balancing>
    </default>

    <readonly>
        <max_memory_usage>10000000000</max_memory_usage>
        <use_uncompressed_cache>0</use_uncompressed_cache>
        <load_balancing>random</load_balancing>
        <readonly>1</readonly>
    </readonly>

</profiles>

<quotas>
    <!-- Name of quota. -->
    <default>
        <interval>
            <duration>3600</duration>
            <queries>0</queries>
            <errors>0</errors>
            <result_rows>0</result_rows>
            <read_rows>0</read_rows>
            <execution_time>0</execution_time>
        </interval>
    </default>
</quotas>
```

http://jackpgao.github.io/2017/11/20/ClickHouse-User-Password/

配置文件：

user.xml

```xml
<users>
    <default>
        <password_sha256_hex>967f3bf355dddfabfca1c9f5cab39352b2ec1cd0b05f9e1e6b8f629705fe7d6e</password_sha256_hex>
        <networks incl="networks" replace="replace">
            <ip>::/0</ip>
        </networks>
        <profile>default</profile>
        <quota>default</quota>
    </default>

    <ck>
        <password_sha256_hex>967f3bf355dddfabfca1c9f5cab39352b2ec1cd0b05f9e1e6b8f629705fe7d6e</password_sha256_hex>
        <networks incl="networks" replace="replace">
            <ip>::/0</ip>
        </networks>
        <profile>readonly</profile>
        <quota>default</quota>
    </ck>
</users>
```

```bash
PASSWORD=$(base64 < /dev/urandom | head -c8);
echo "$PASSWORD"; echo -n "$PASSWORD" | sha256sum | tr -d '-'
```

http://jackpgao.github.io/2017/11/20/ClickHouse-User-Password/

ClickHouse的问题：

1. insert into xx (a, b, c) values ('a', 'b', 'c') 只能是单引号

2. 如果是int插入的是string，报错（不确定是否有类似SQL_MODE的参数）

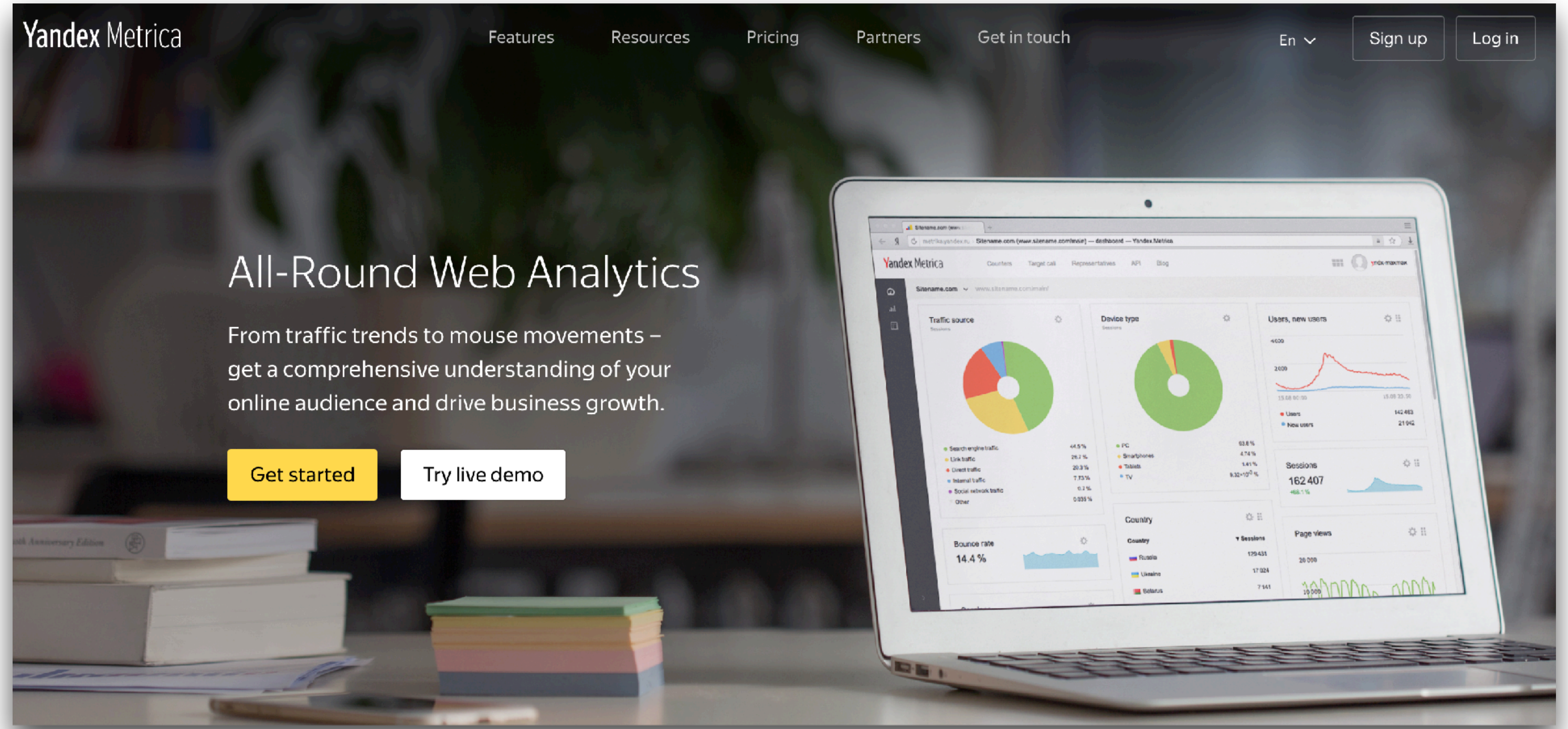
3. 删除只支持到月纬度的分区

4. 改造官方的启动脚本，不要用root直接启动

谁在用

Yandex Metrica
Features
Resources
Pricing
Partners
Get in touch
En
Sign up
Log in
All-Round Web Analytics
From traffic trends to mouse movements – get a comprehensive understanding of your online audience and drive business growth.
Get started
Try live demo

谁在用



374 servers

store over 20.3 trillion rows

17 PB uncompressed data TSV

2 PB without counting duplication and replication

谁在用

CERN

欧洲原子能研究机构

store and process metadata on 10 billion events with over 1000 attributes per event

CLOUDFLARE

《How Cloudflare analyzes 1M DNS queries per second》

Multi-tenant ClickHouse cluster

33	8M+	4GB+	2PB+
Nodes	Row Insertion/s	Insertion Throughput/s	Raid-0 Spinning Disks

Tinkoff Bank

Innovative provider of online retail financial services in Russia

俄罗斯互联网金融

谁在用

- 做存储的公司
- Python驱动：infi.clickhouse_orm

- CARTO (formerly CartoDB) is a Software as a Service (SaaS) cloud computing platform that provides GIS and web mapping tools for display in a web browser.

谁在用

- Altinity is the leading service provider for ClickHouse
- 高管来自Percona、ClickHouse作者

- Altinity Provides Cloud Version of ClickHouse on Kodiak Data MemCloud™ 提供云化的ClickHouse服务

- 开源MySQL实时同步数据到ClickHouse工具

Altinity

https://www.altinity.com/blog/

谁在用

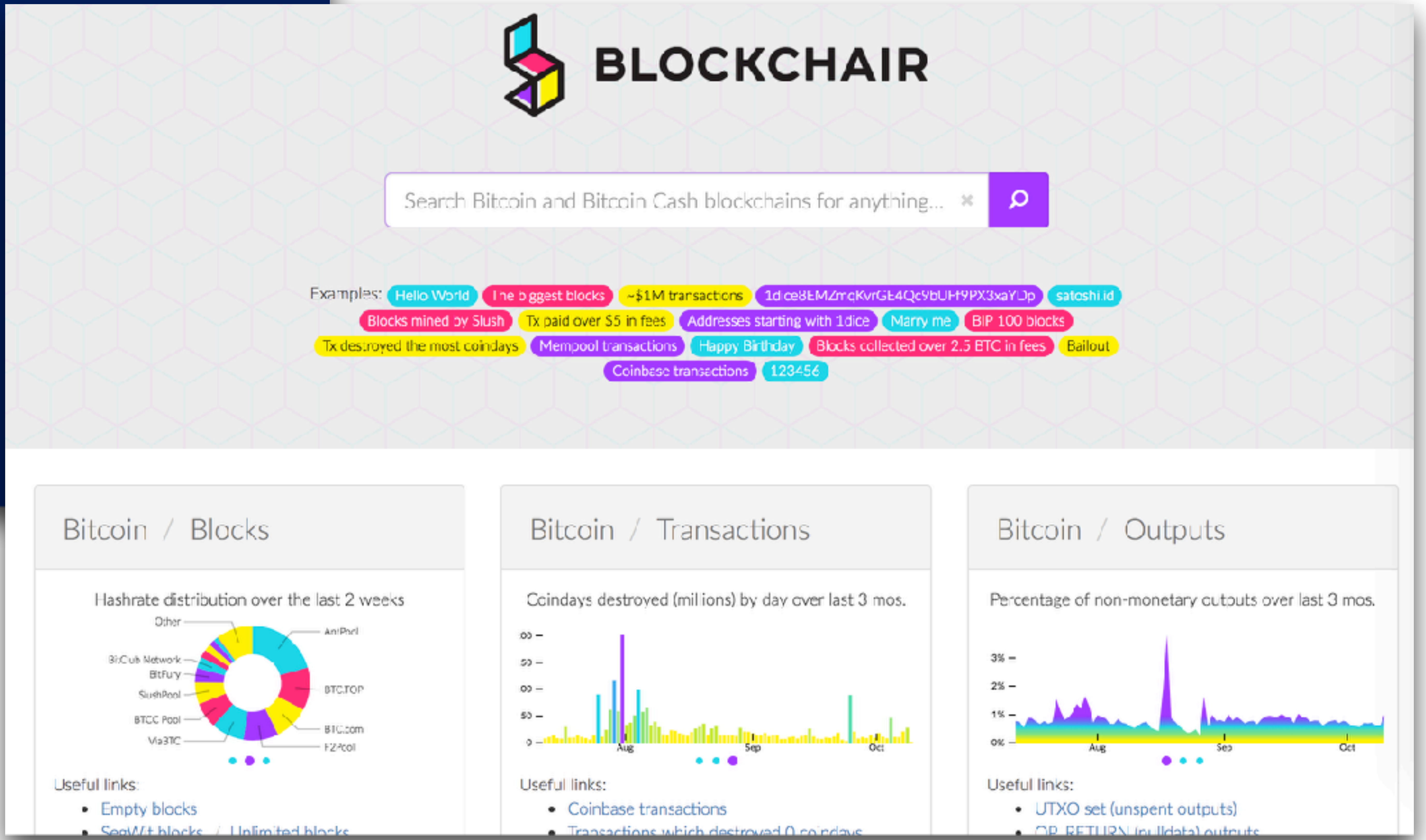
PMM Roadmap

- Alerting
- QAN for MongoDB
- MySQL -> ClickHouse for QAN datastore
- Plugins and Integrations
- Long term metrics storage (past 30 days)
- One-click ticket submission*
- Standardised data collection for tickets*

Any feedback of what you'd like to see in PMM?

* for Percona Subscribers (Customers) only

PERCONA
Monitoring and Management



Blockchair is a blockchain search and analytics engine or you can also say it's a blockchain explorer on steroids.

谁在用

IPINYOU 品友互动

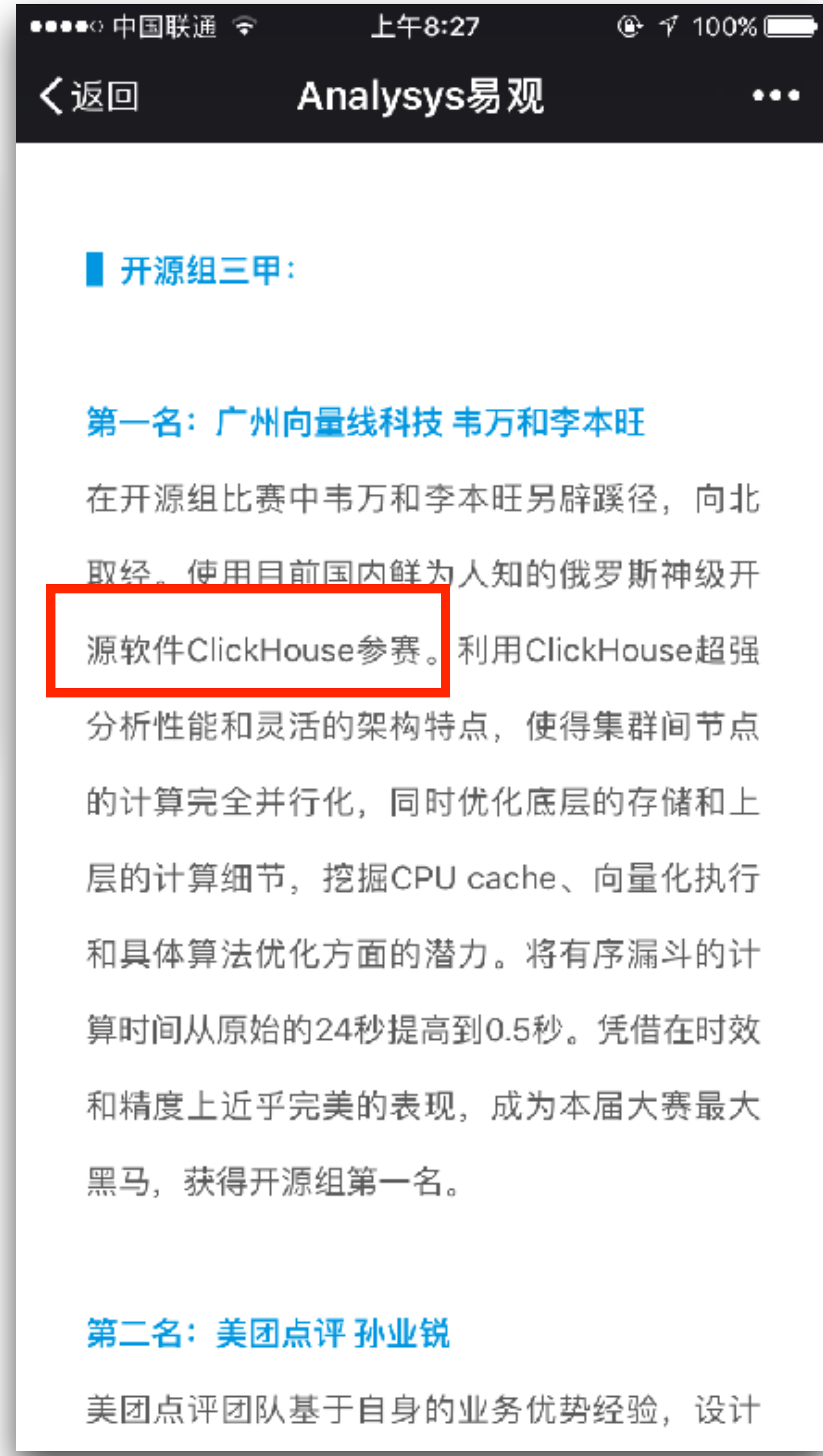
人群定向 智慧传播

ONE APM

策导信息为金风科技设计方案

使用21台ClickHouse

存储风力发电机监控数据



开源组三甲：

第一名：广州向量线科技 韦万和李本旺

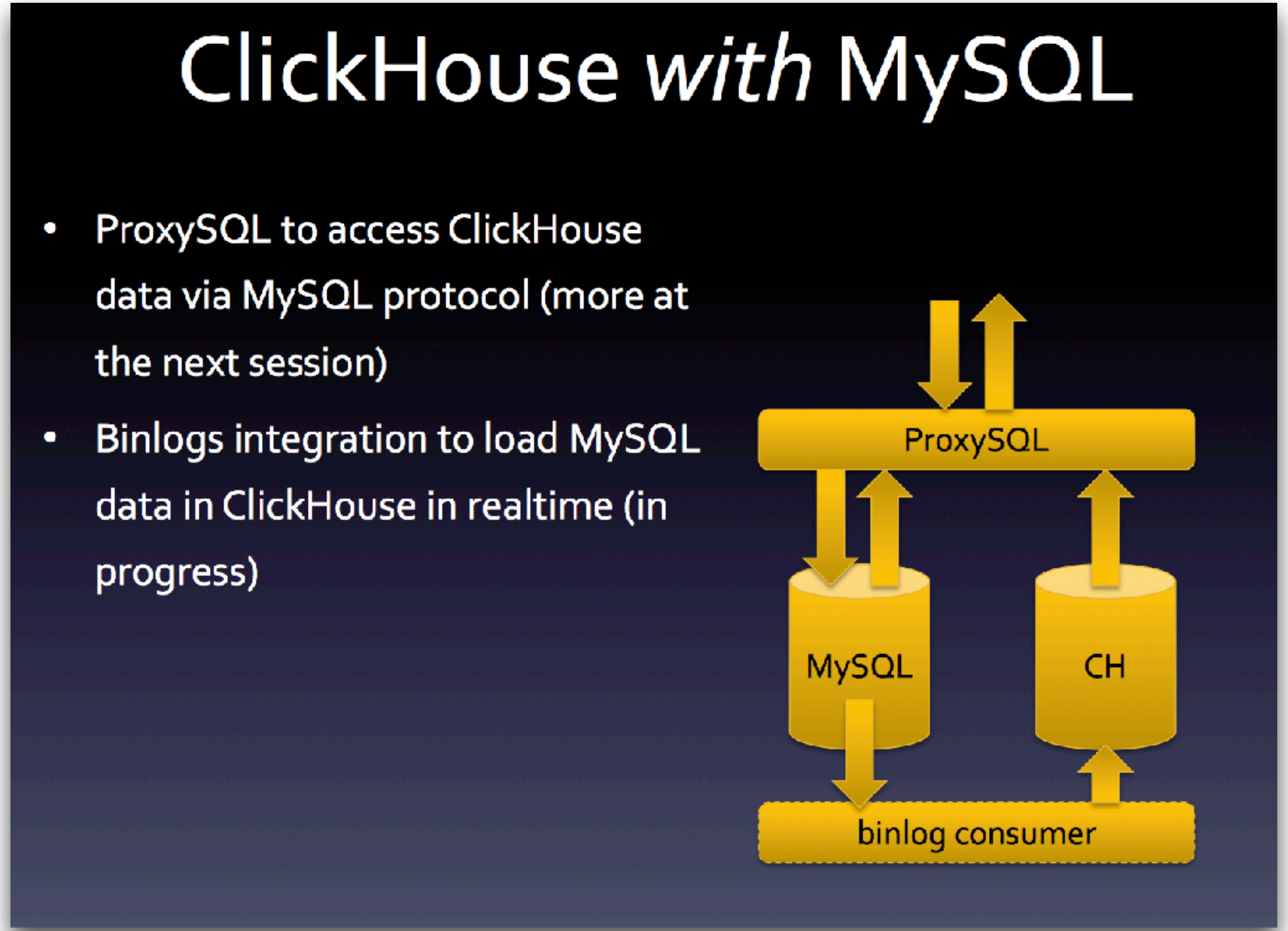
在开源组比赛中韦万和李本旺另辟蹊径，向北取经。使用目前国内鲜为人知的俄罗斯神级开源软件ClickHouse参赛。利用ClickHouse超强分析性能和灵活的架构特点，使得集群间节点的计算完全并行化，同时优化底层的存储和上层的计算细节，挖掘CPU cache、向量化执行和具体算法优化方面的潜力。将有序漏斗的计算时间从原始的24秒提高到0.5秒。凭借在时效和精度上近乎完美的表现，成为本届大赛最大黑马，获得开源组第一名。

第二名：美团点评 孙业锐

美团点评团队基于自身的业务优势经验，设计

谁在关注

ProxySQL支持ClickHouse作为后端，使用MySQL协议访问

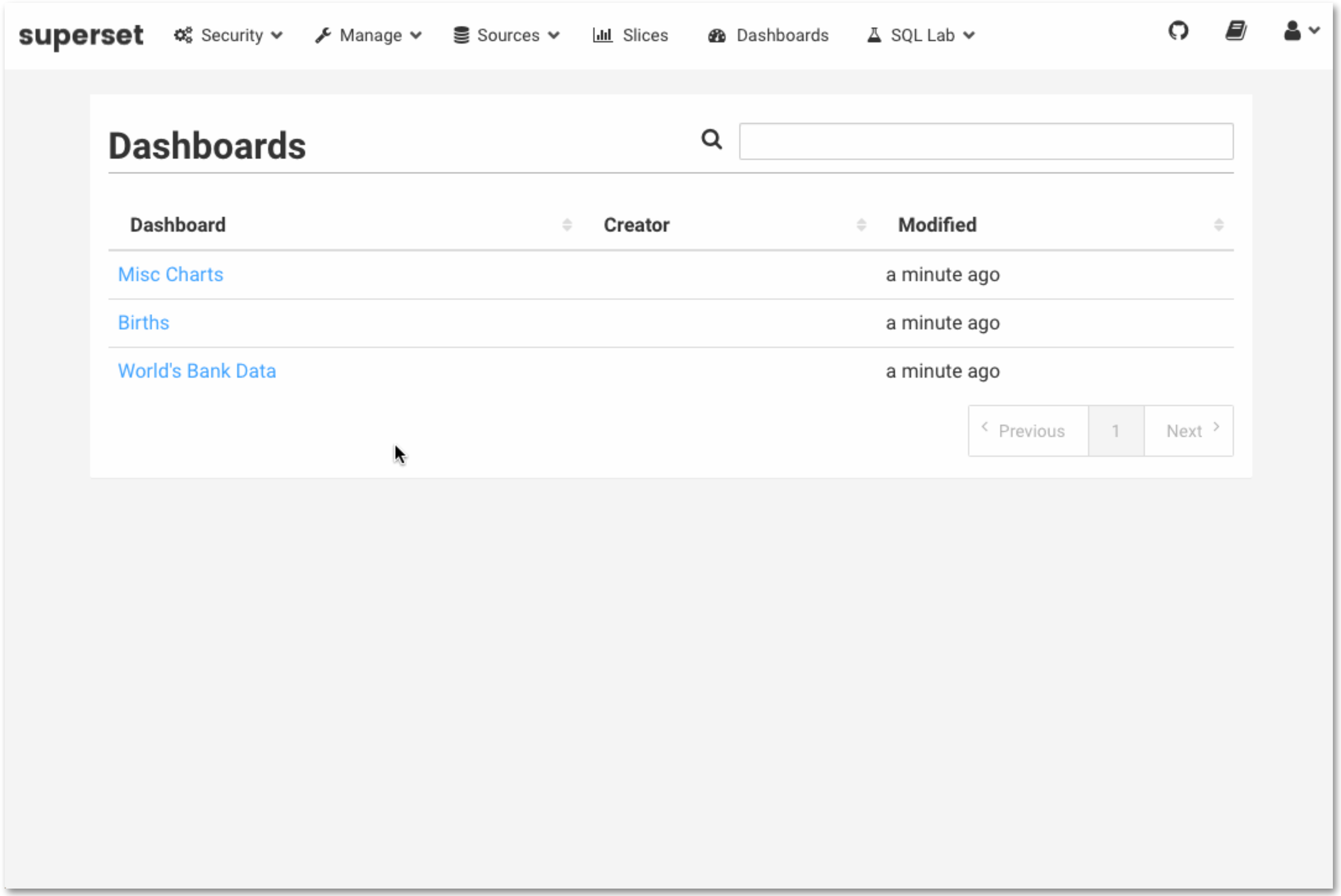


http://www.proxysql.com/blog/proxysql-143-clickhouse

我浪怎么用?

搭配

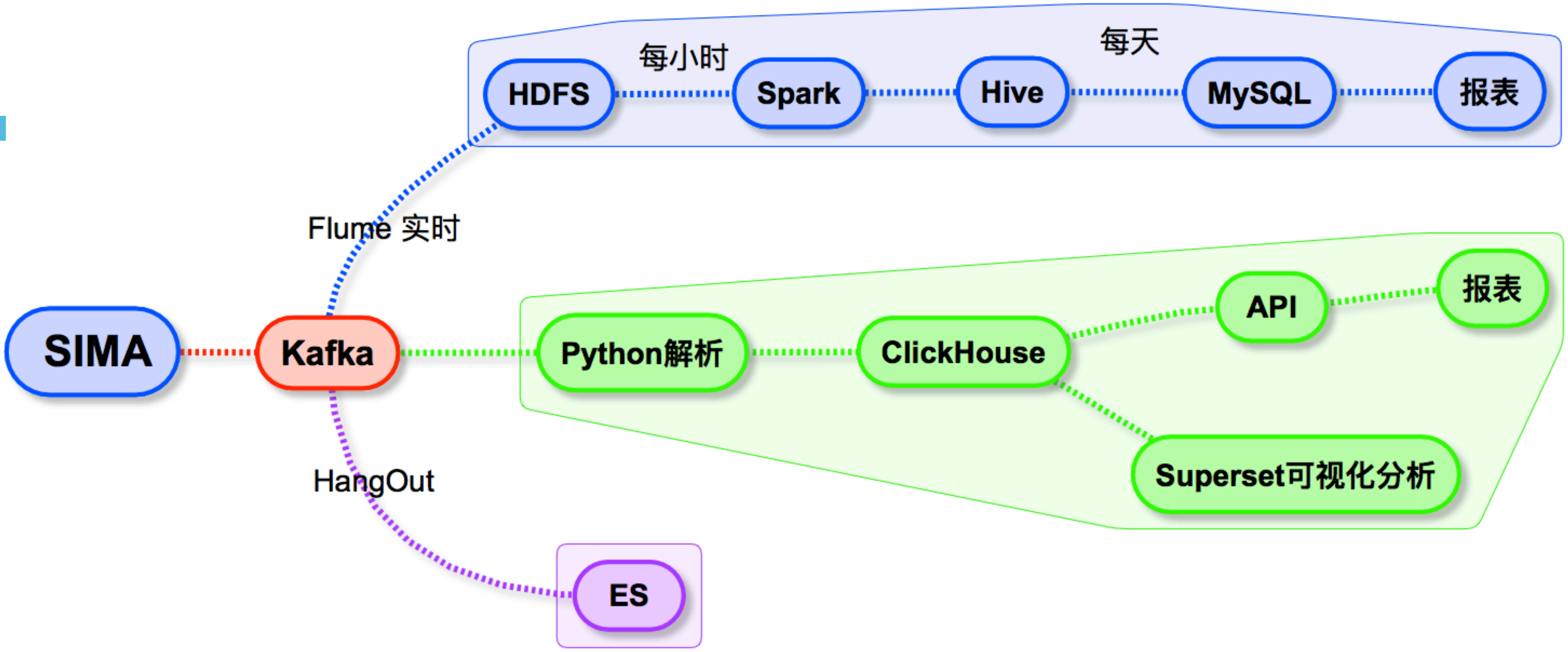
- Superset



- Airbnb开源Python项目
- Apache孵化
- 快速实现可视化
- 完美对接ClickHouse，
  方便分析师进行问题排查与分析

案例

- 某APP性能监控



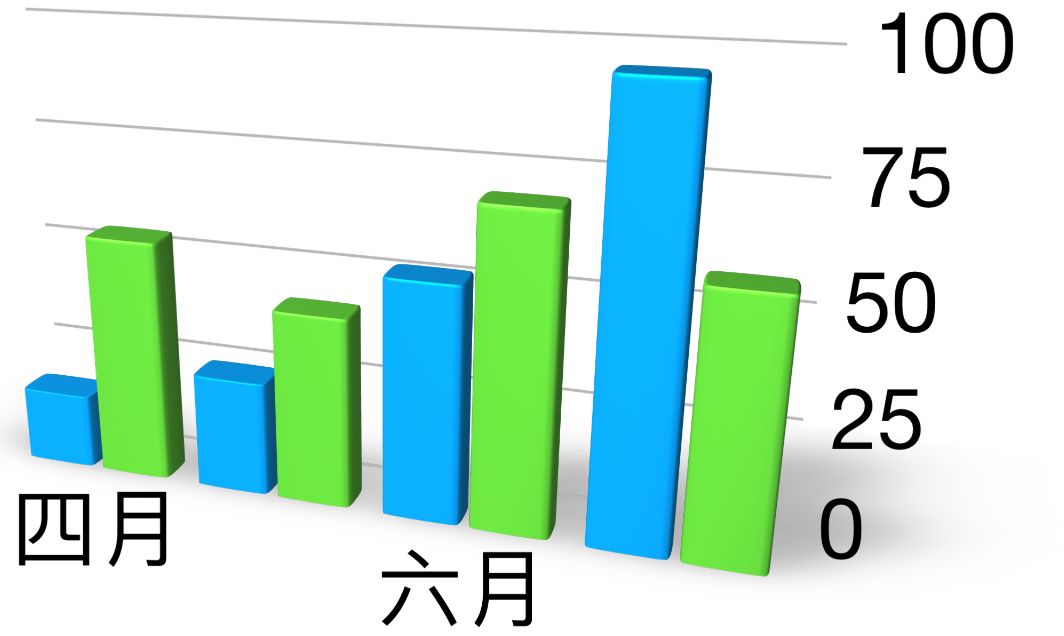
每小时
每天
HDFS
Spark
Hive
MySQL
报表
Flume 实时
SIMA
Kafka
Python解析
ClickHouse
API
报表
Superset可视化分析
HangOut
ES

案例

kafka

- Python
- Hangout

- ClickHouse



- 图表、邮件、报警等
- Superset可视化工作台

- ES

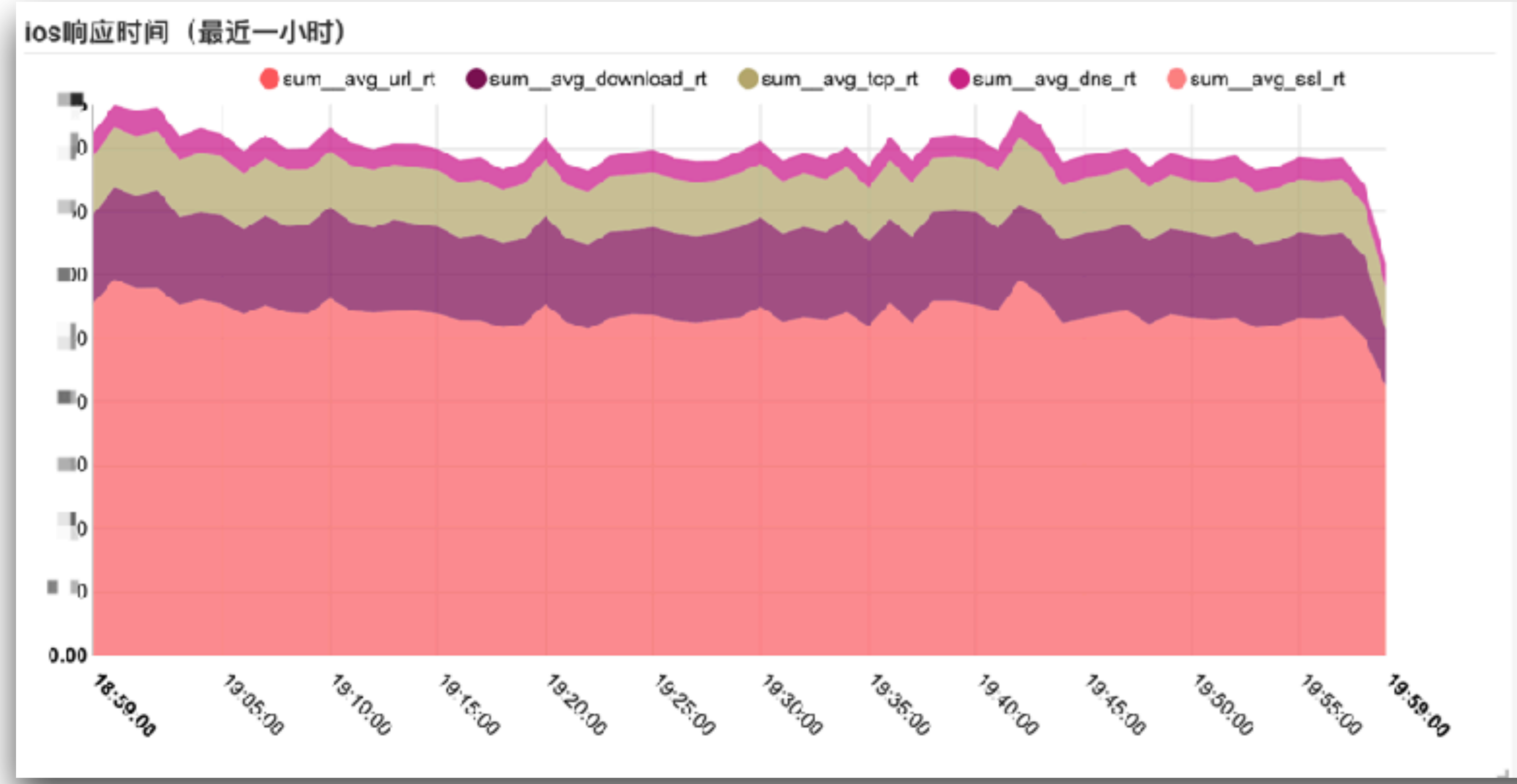
kibana

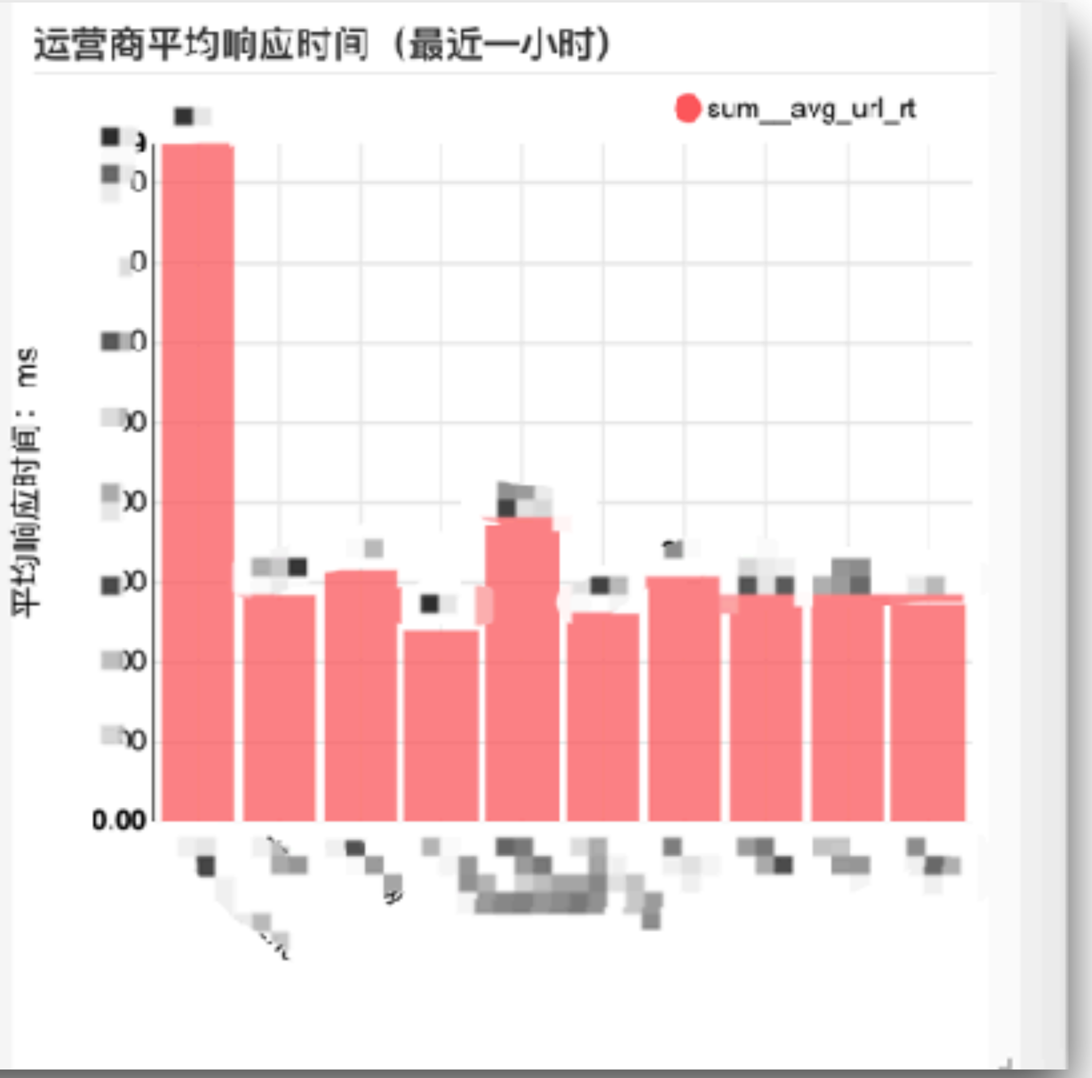
- 问题排查，Trace

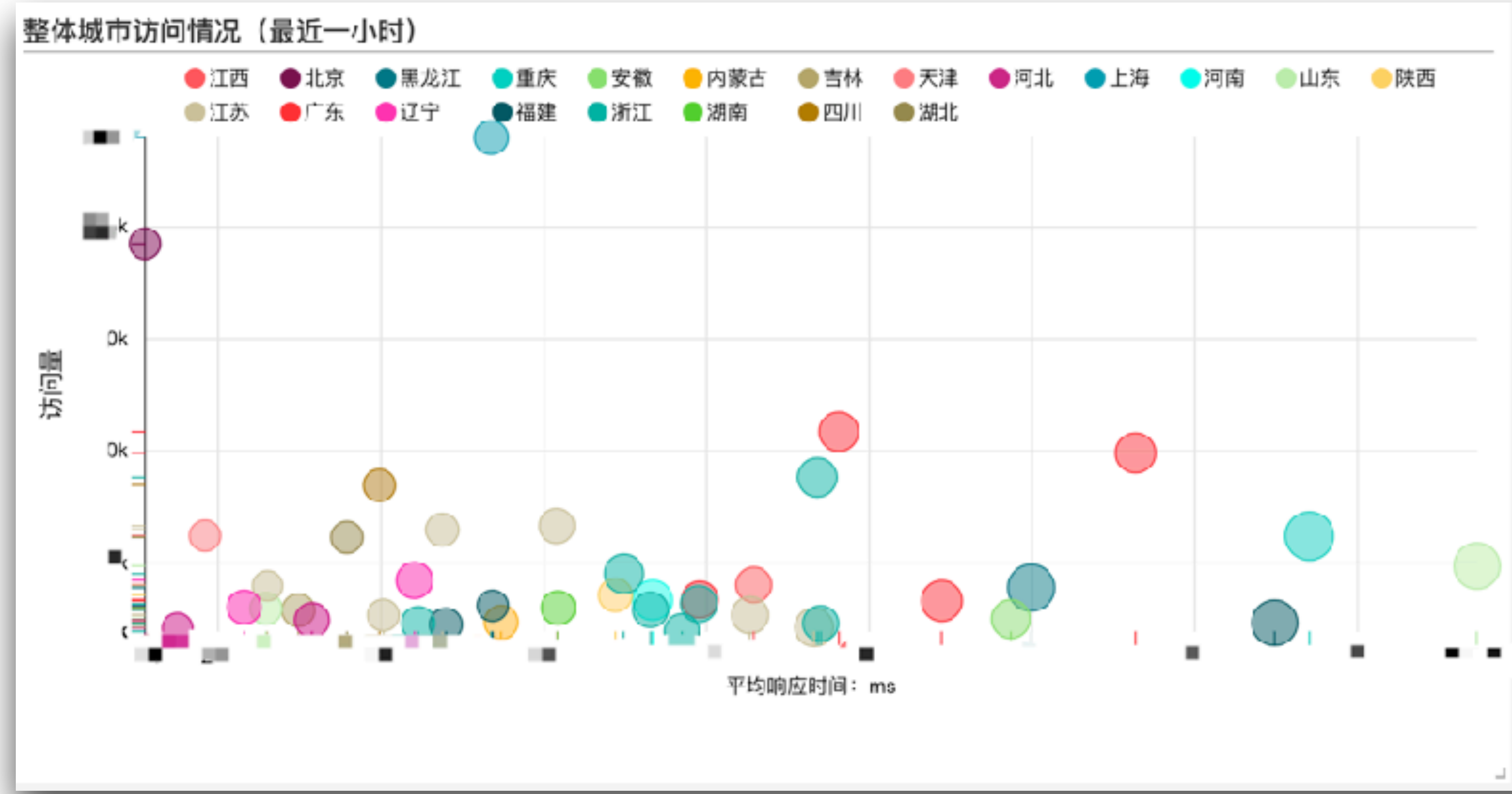
- 数据处理链路短
- 数据实时可见，及时Trace
- 如何快速数据变现

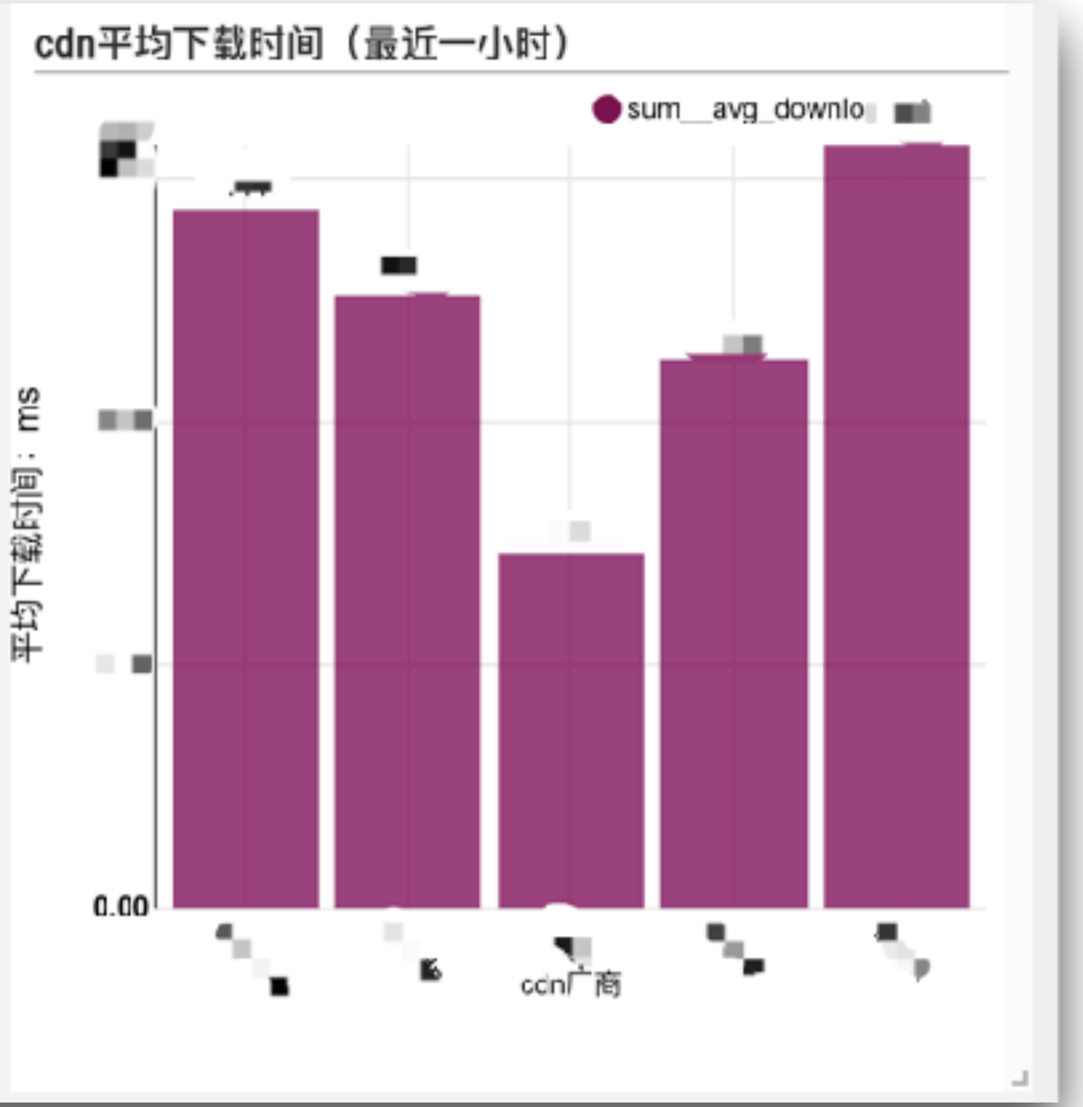
案例

- 某APP性能监控


ios响应时间（最近一小时）
sum__avg_url_rt
sum__avg_download_rt
sum__avg_tcp_rt
sum__avg_dns_rt
sum__avg_ssl_rt
0.00
18:50:00
19:05:00
19:10:00
19:15:00
19:20:00
19:25:00
19:30:00
19:35:00
19:40:00
19:45:00
19:50:00
19:55:00
19:59:00

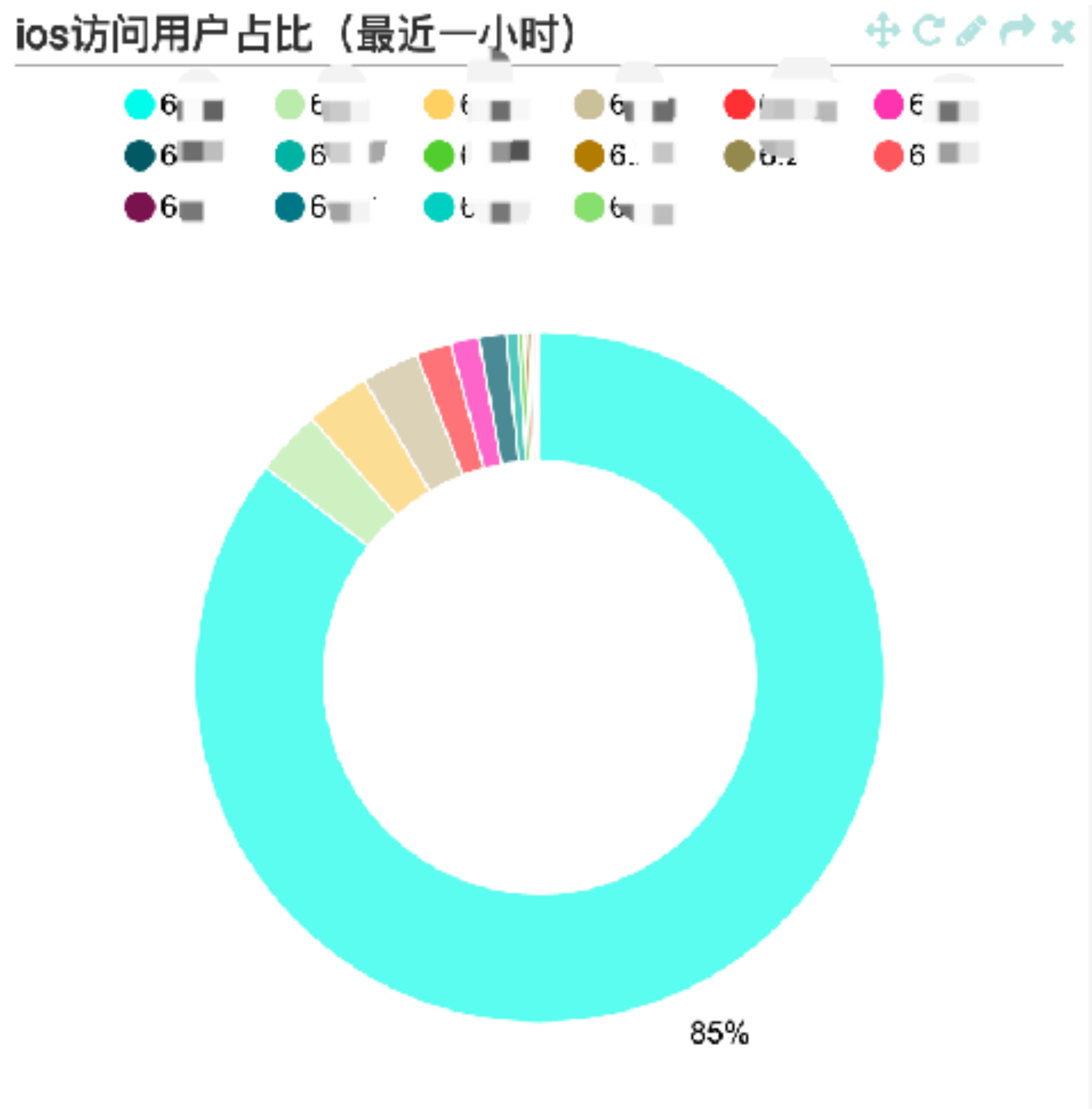

运营商平均响应时间（最近一小时）
sum__avg_url_rt
平均响应时间：ms
0.00


整体城市访问情况（最近一小时）
江西
北京
黑龙江
重庆
安徽
内蒙古
吉林
天津
河北
上海
河南
山东
陕西
江苏
广东
辽宁
福建
浙江
湖南
四川
湖北
访问量
平均响应时间：ms

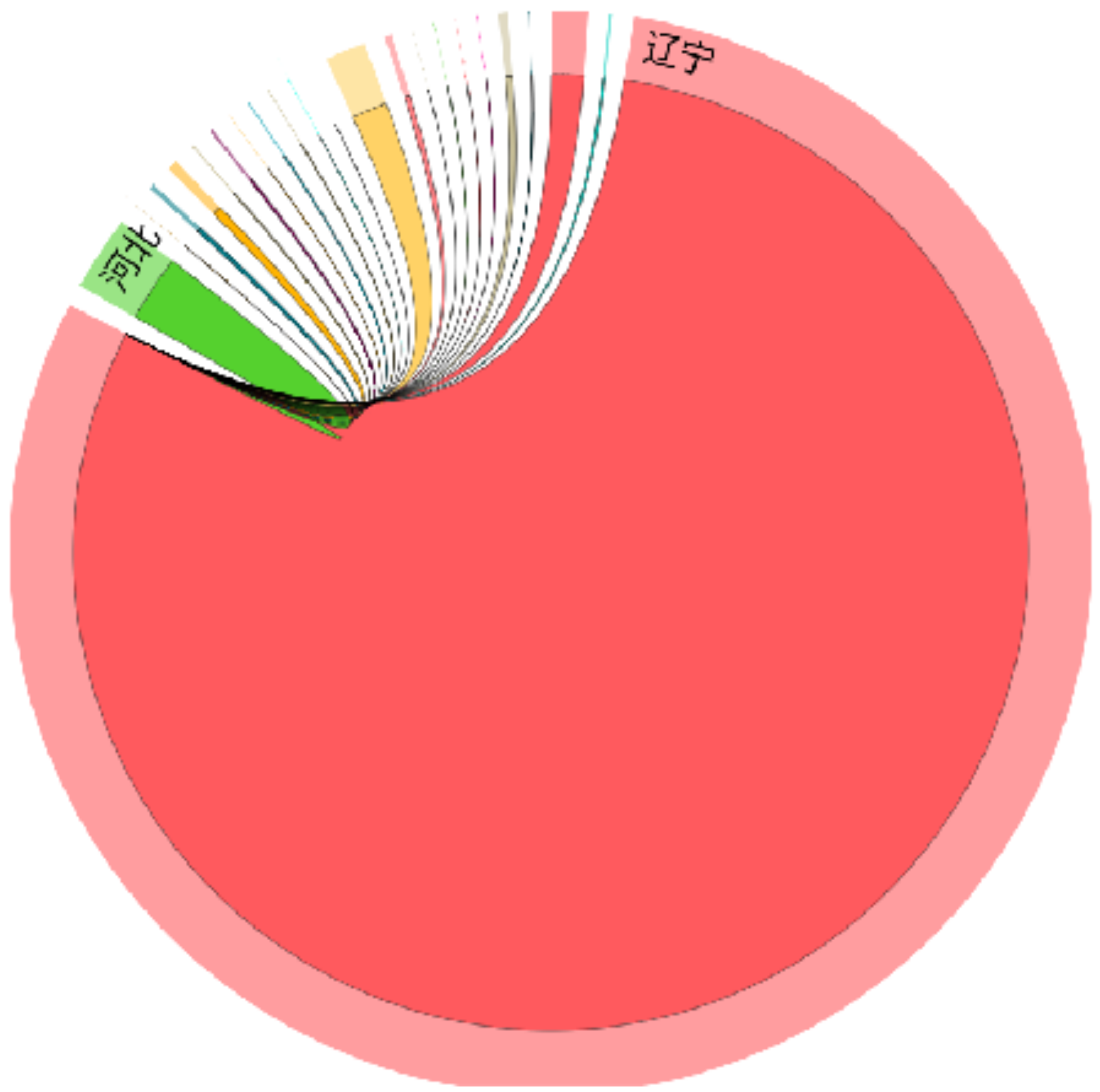

cdn平均下载时间（最近一小时）
sum__avg_downlo
平均下载时间：ms
0.00
cdn厂商

案例

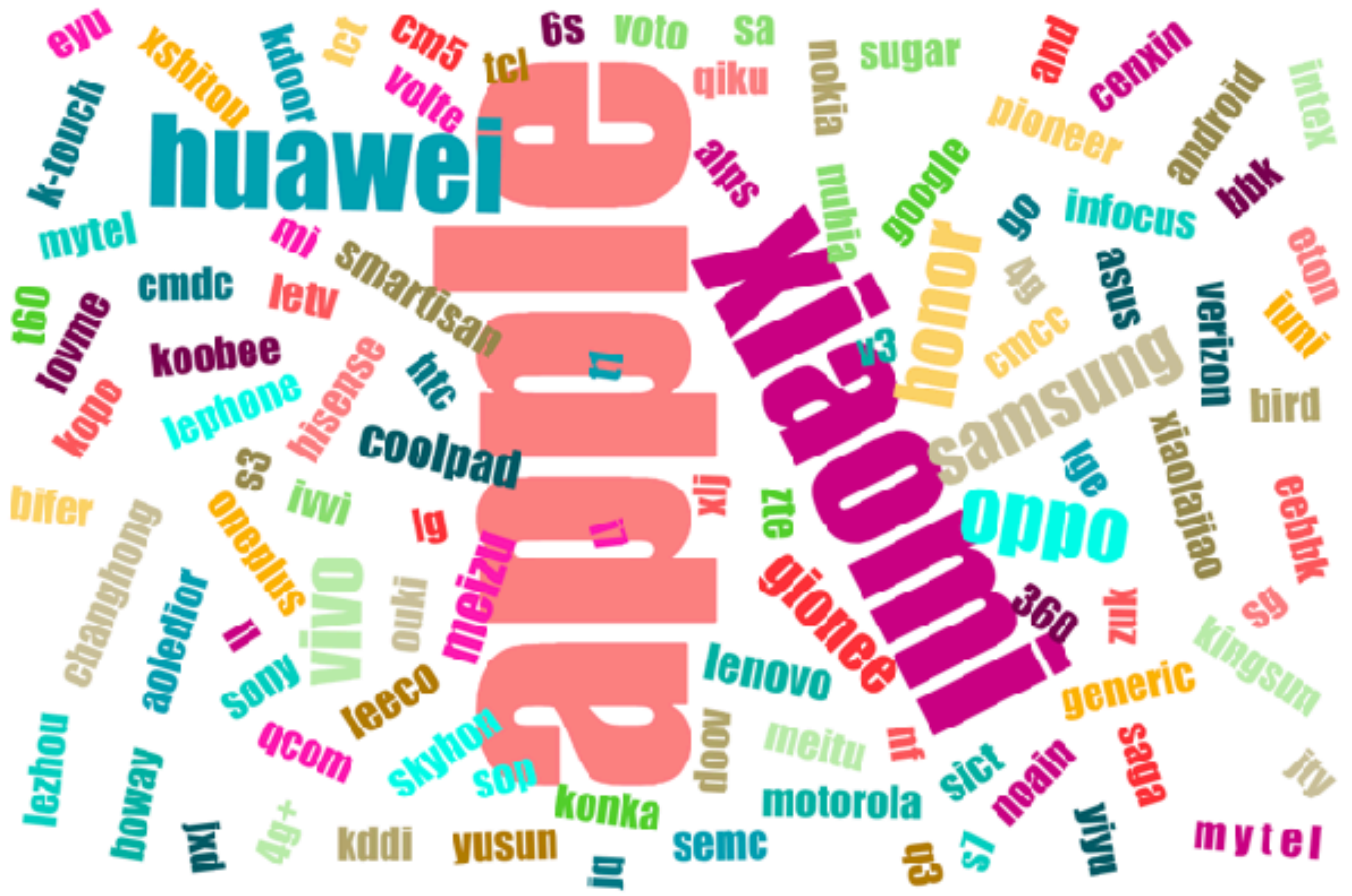
- 某APP性能监控



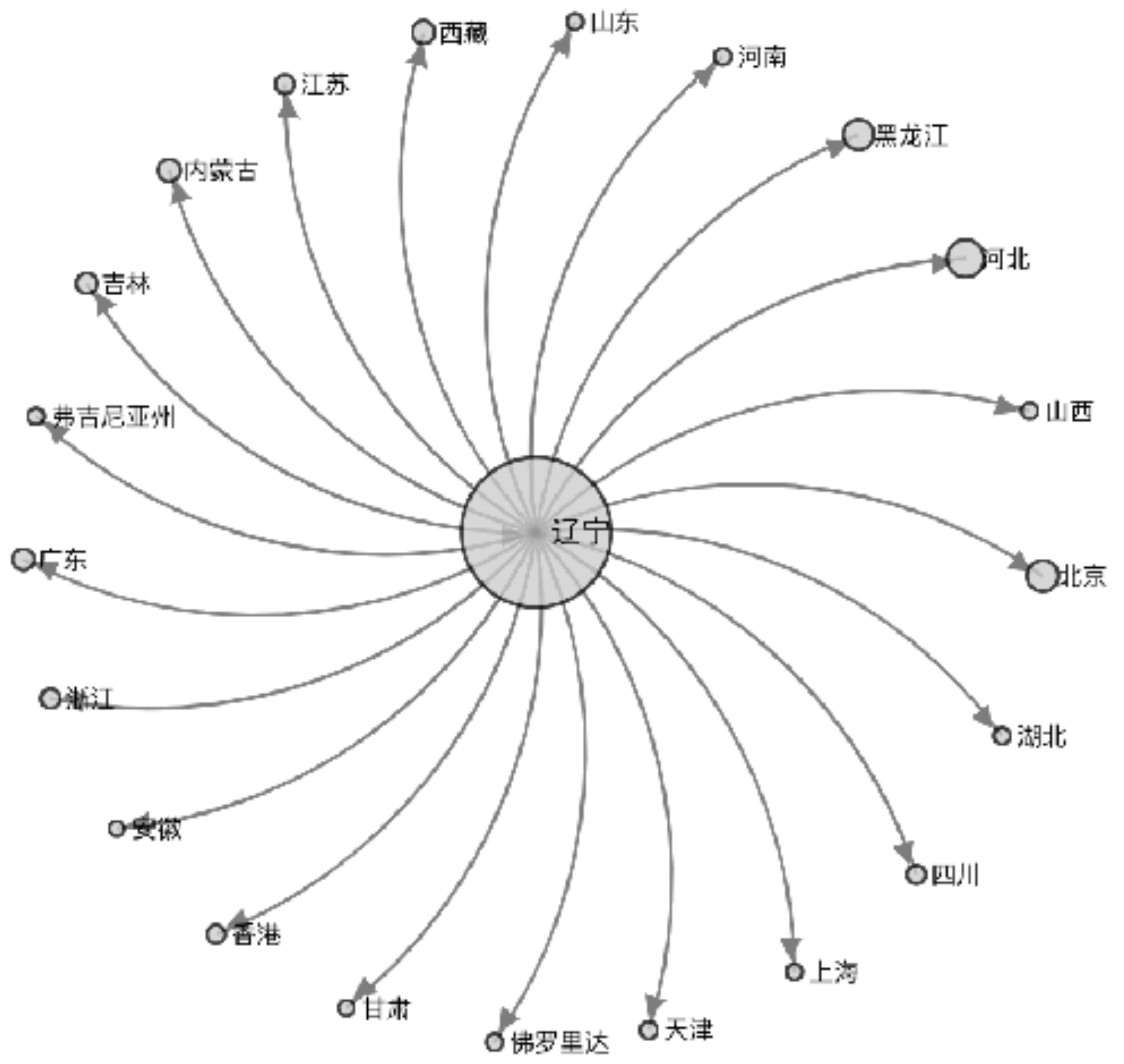
ios访问用户占比（最近一小时）
85%



辽宁
河北



apple
xiaomi
huawei
samsung
oppo
honor
vivo
gionee
lenovo
coolpad
meizu
motorola

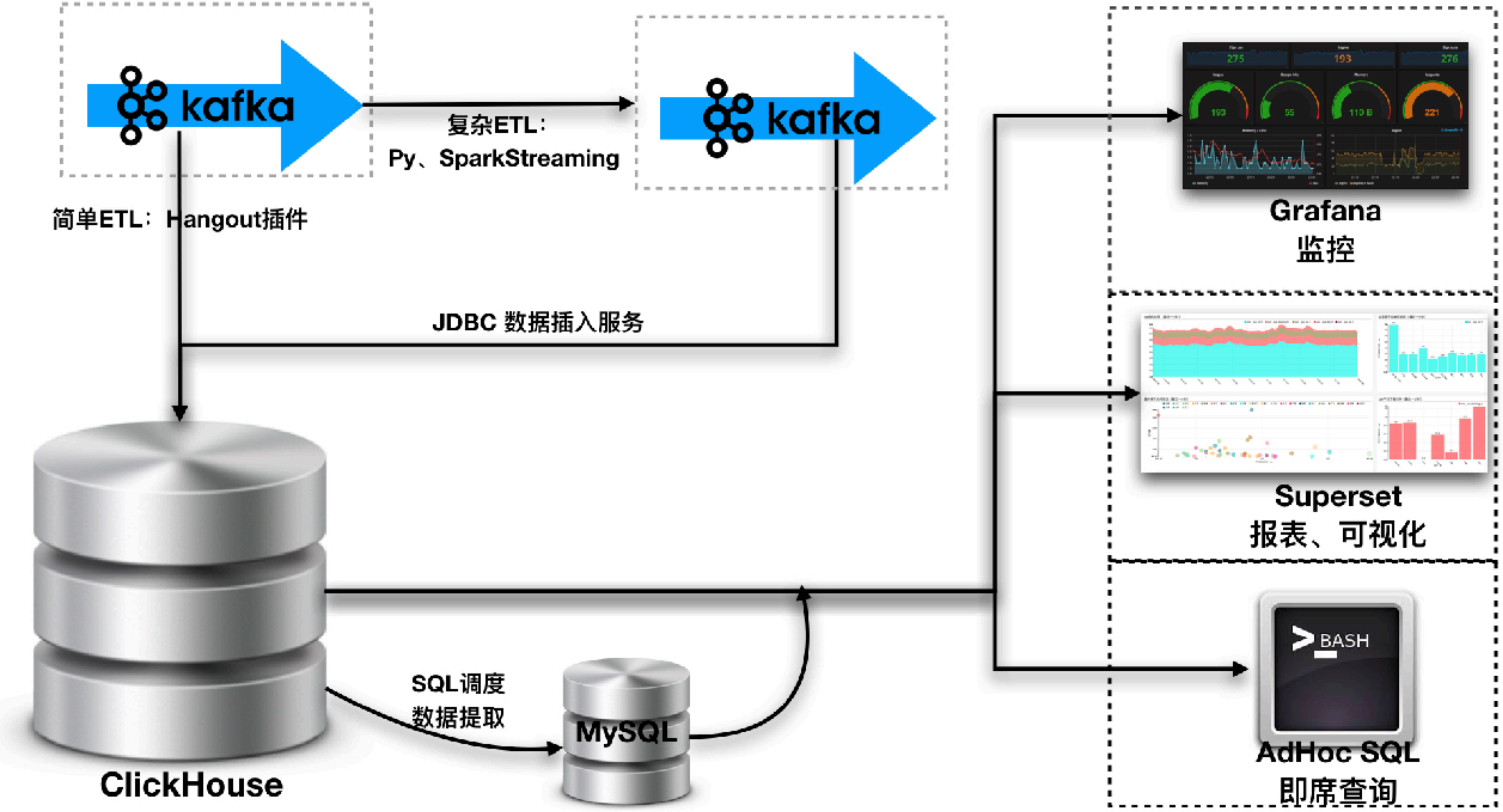


辽宁
西藏
山东
河南
江苏
黑龙江
内蒙古
河北
吉林
山西
弗吉尼亚州
广东
北京
浙江
湖北
安徽
四川
香港
上海
甘肃
佛罗里达
天津

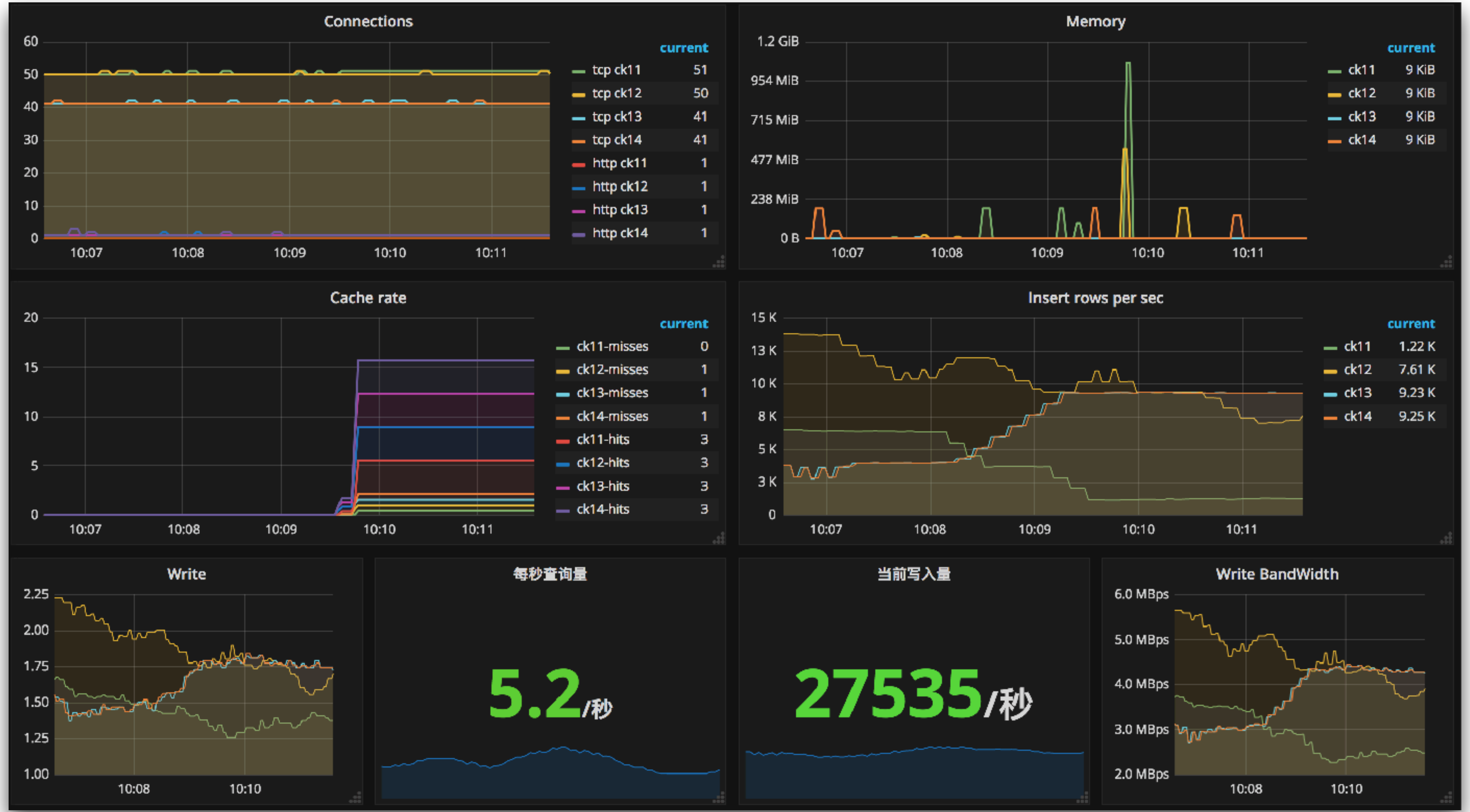
案例

数据架构

- Hangout Plug-in：
  - 支持简单ETL操作，
  - 支持Kafka直达ClickHouse
- SQL调度组件：
  - 自动调度报表任务SQL，
  - 支持数据重跑等功能
- 数据入库组件：
  - 复杂ETL操作，清洗操作不做入库，
  - 重新丢入Kafka，异步批量入库



CK监控



Prometheus

+

ClickHouse exporter

+

Grafana

https://github.com/f1yegor/clickhouse_exporter

最佳实践

运维

1. CPU：多核优于主频
   (SSE 4.2 instruction set need)
   开启超线程、性能模式

2. 小磁盘多机器>大磁盘少机器

3. Raid-10 is better
   If Raid5/6/50, increase stripe_cache_size

4. 内存：越大越好，留给page cache
   禁用swap/透明大页/NUMA

5. CentOS7/Ext4/复制带宽问题

使用

1. batch insert 2K 起步
   过多并发查询，不是它的菜

2. 用域名写本地表，读分布式表

3. 如果是Docker，注意修改时区

4. clickhouse-client在Docker里，
   中文乱码

5. 拒绝 select *

6. 无Decimal，乘以倍率，用Uint64存

总结

使用场景

前提：对事务无要求 无update操作 对响应时间有要求

特征：体量大 结构化

接口：SQL Http API Py PHP R

案例：日志数据 广告曝光 IoT 监控数据

其他选项?

对比

综合

MySQL：防止撕逼、解放DBA的利器，从此告别容量和慢查问题

预处理类，Druid/Kylin等：保留原始数据，防止预先设定不满足需求

ES：见后文

HDFS生态：简单、易用、查询快，但是，规模稳定性有待验证

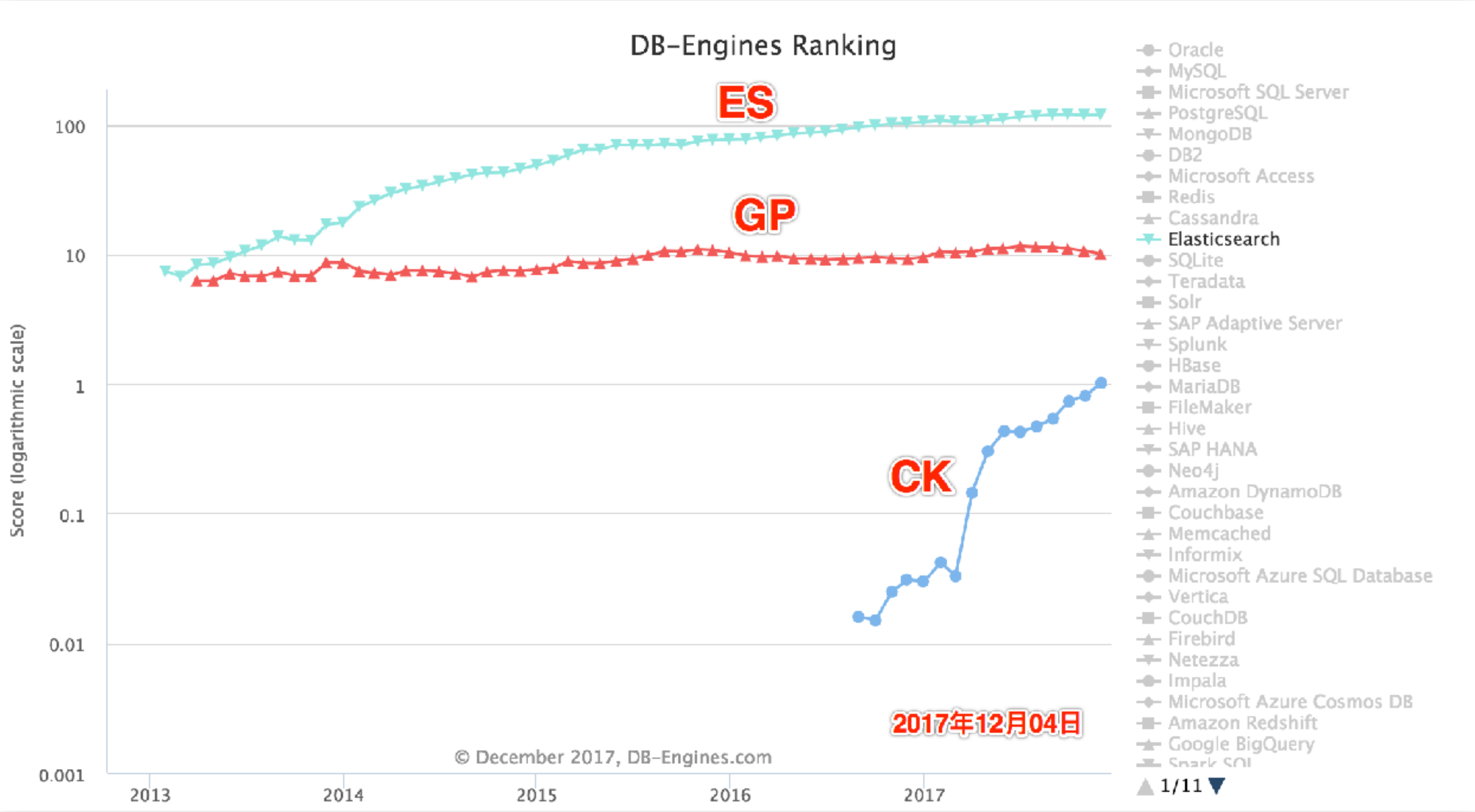
对比-ES

对比ES

对比项	ES	ClickHouse
数据接入	插件丰富：hangout/logstash/filebeats 无需特别指明字段类型，兼容性好	需要自己开发：JDBC、clickhouse-client、Python导入 **Hangout to ClickHouse插件**
查询	原生查询方式不灵活 SQL插件复杂度有限 大范围查询性能差	支持**复杂查询** **高级函数**多 支持**连表查询**
扩展性	旗鼓相当，ES略轻松，ClickHouse需要管理表维度	
其他	原生API **社区健壮**	原生API 社区正在发展
语言	Java/Python	Java/Python/R/PHP

对比-ES

行业排名



DB-Engines Ranking
ES
GP
CK
2017年12月04日
Score (logarithmic scale)
100
10
1
0.1
0.01
0.001
2013
2014
2015
2016
2017
© December 2017, DB-Engines.com
Oracle
MySQL
Microsoft SQL Server
PostgreSQL
MongoDB
DB2
Microsoft Access
Redis
Cassandra
Elasticsearch
SQLite
Teradata
Solr
SAP Adaptive Server
Splunk
HBase
MariaDB
FileMaker
Hive
SAP HANA
Neo4j
Amazon DynamoDB
Couchbase
Memcached
Informix
Microsoft Azure SQL Database
Vertica
CouchDB
Firebird
Netezza
Impala
Microsoft Azure Cosmos DB
Amazon Redshift
Google BigQuery
1/11

对比-GPU产品

MAPD

- GPU Database for Fast, Interactive Visual Analytics

GPU产品

kinetica

- GPU-accelerated analytics database for real-time insights on large and streaming datasets

BLAZINGDB

- High Performance GPU Database for Big Data SQL

brytlyt

- World's most **advanced** GPU based PostgreSQL Database

http://tech.marksblogg.com/benchmarks.html

对比-GPU产品

MAPD

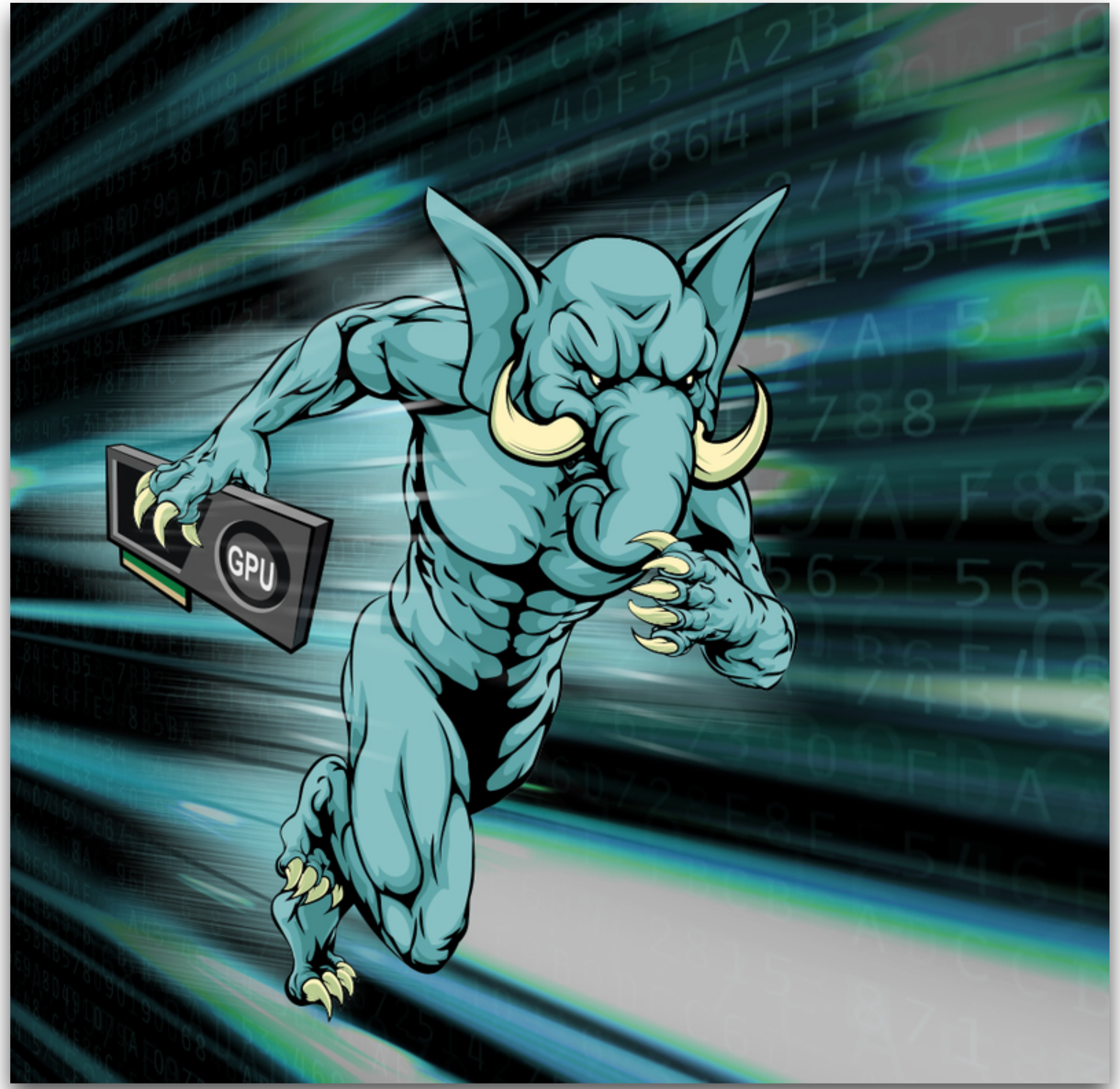
74x to 3,500x faster than CPU DBs.

The table is sorted by the fastest time query 1 finished in (measured in seconds).

Query 1	Query 2	Query 3	Query 4	Setup
0.021	0.053	0.165	0.51	MapD & 8 Nvidia Pascal Titan Xs
0.027	0.083	0.163	0.891	MapD & 8 Nvidia Tesla K80s
0.028	0.2	0.237	0.578	MapD & 4-node g2.8xlarge cluster
0.036	0.131	0.439	0.964	MapD & 4 Nvidia Titan Xs
0.051	0.146	0.047	0.794	kdb+/q & 4 Intel Xeon Phi 7210 CPUs
1.034	3.058	5.354	12.748	ClickHouse, Intel Core i5 4670K
1.56	1.25	2.25	2.97	Redshift, 6-node ds2.8xlarge cluster
2	2	1	3	BigQuery
4	4	10	21	Presto, 50-node n1-standard-4 cluster
6.41	6.19	6.09	6.63	Amazon Athena
8.1	18.18	n/a	n/a	Elasticsearch (heavily tuned)
10.19	8.134	19.624	85.942	Spark 2.1, 11 x m3.xlarge cluster w/ HDFS
11	10	21	31	Presto, 10-node n1-standard-4 cluster
14.389	32.148	33.448	67.312	Vertica, Intel Core i5 4670K
34.48	63.3	n/a	b/a	Elasticsearch (lightly tuned)
35	39	64	81	Presto, 5-node m3.xlarge cluster w/ HDFS
43	45	27	44	Presto, 50-node m3.xlarge cluster w/ S3
152	175	235	368	PostgreSQL 9.5 & cstore_fdw
264	313	620	961	Spark 1.6, 5-node m3.xlarge cluster w/ S3

对比-GPU产品

- PG-Strom is an extension designed for PostgreSQL v9.5 or later, to off-load a part of CPU intensive workloads to GPU (Graphic Processor Unit) devices, and execute them in parallel asynchronously.



对比-开源与商业

Commercial solutions – fast and expensive

- Vertica
- RedShift
- Teradata
- Etc
- The cost scales with your data

商业产品

开源产品

Open Source: somewhat slow, sometime buggy. But free

- InfiniDB (now MariaDB ColumnStore)
- InfoBright
- GreenPlum (started as commerical)
- Hadoop systems
- Apache Spark

对比-开源与商业

性能与成本的均衡

结缘，

“那年我还是个DBA，饱受业务复杂查询，也就是OLAP之苦”

https://www.percona.com/

不怕有坑?

不试试

怎么知道好用不好用

Summary

Try it now!

大容量结构化的数据

需要SQL

快速实现聚合、可视化

- But，如果，
- 不好用，别撕我~

资源推荐：

JackpGao

JackGao
北京 海淀

扫一扫上面的二维码图案，加我微信

gaopeng4@staff.sina.com.cn

1. 官方文档 ☆☆☆ 结构清晰，文档清楚，但是例子太少
2. Percona ClickHouse Blog ☆☆☆☆☆
3. github issue & ClickHouse Google Group
4. ClickHouse服务提供商Altinity ☆☆☆☆☆
5. 官方Meetup PPT ☆☆☆☆☆
6. Altinity提供的rpm包 ☆☆☆☆☆
7. 官方运维建议
8. 个人推荐PPT合集：百度网盘，密码yv72