# clickhouse集群方案调研报告

## 1.问题

目前集中域管平台使用到了clickhouse作为数据库存储组件，主要存储日志，以及报表相关的数据，主要用途为olap。

平台使用clickhouse所处理的数据量相对于mysql所处理的数据量更大。目前集中域管平台的mysql有主从和mha高可用的部署支持，一定程度上解决了单点故障的问题。

但是clickhouse目前仍然是单点部署，如果该clickhouse节点出现问题，则整个平台的日志服务和报表服务将会失效，对应的接口将会超时和报错，甚至由此引发其他的一系列连锁问题，因此增加clickhouse集群部署方案迫在眉睫。

## 2.现状

目前的集中域管平台，包括单机部署和K8S高可用部署两种方案，其使用到的clickhouse数据库均为单一节点部署，这意味着一旦该clickhouse节点出现宕机等异常情况，将会严重影响整个平台所提供的正常服务。

为了解决上述clickhouse单点部署存在的问题，考虑实现clickhouse的集群部署，将clickHouse的服务拓扑由单节点延伸到多个节点，当clickhouse集群中的一个或若干节点出现故障后，整个平台仍然能提供的正常服务。

Mergetree是clickhouse的最常用最核心的表引擎，集中域管平台使用的也是该引擎，ReplicatedMergeTree是MergeTree的派生引擎，它在MergeTree的基础上加入了分布式协同的能力，使用了ReplicatedMergeTree复制表系列引擎，便能应用副本的能力。用一种更为直接的方式理解，即使用ReplicatedMergeTree的数据表就是副本。

虽然集群能够有效解决单点故障的问题，同时也意味着集群各个节点之间的通信，以及数据一致性问题需要得到解决。

实际数据存储业务中，虽然clickhouse和mysql的使用形式很像，但在集群部署层面和mysql完全不同：

- clickhouse没有类似mysql主从同步的概念，采用的是副本概念的多主架构；
- clickhouse的数据同步是表级别，不是数据库层面的数据复制与同步；
- 在保证数据一致性方面，clickhouse采用的方式是通过特定的表引擎（ReplicatedMergeTree) 以及集成zookeeper来实现。zookeeper作为外部组件存在，不需要和clickhouse集群部署在一起。

## 3.技术方案一

## 3.1 【副本模式】

**副本模式：**在集群的各个节点上，每个节点都拥有同样的数据，使用副本的主要目的是防止数据丢失，增加数据存储的冗余。

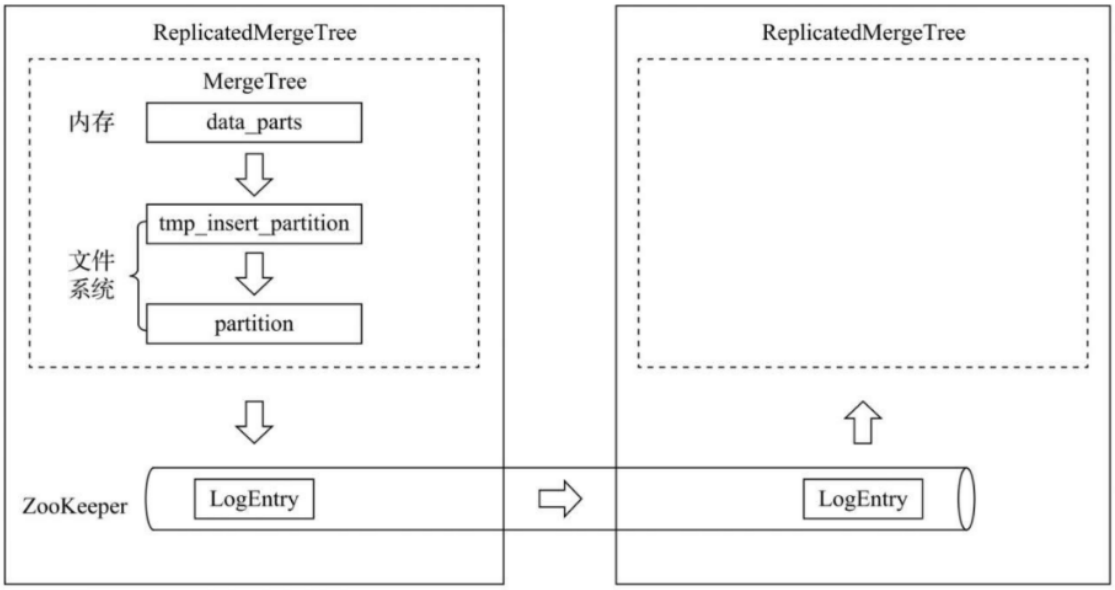
采用副本模式的clickhouse集群uml图如下：

clickhousea，clickhouseb，clickhousec这三个节点共同组成clickhouse集群，分别与zookeeper集群进行关联，client往任意一个clickhouse节点写入数据均能同步到其他另外两个集群节点，同理，client连接到clickhouse集群中的任意一个节点均能读取到完整数据，3个节点互为副本。

**副本模式是一种多主架构。**

## 3.2 关键技术

前文有介绍ReplicatedMergeTree是MergeTree的派生引擎，它在MergeTree的基础上加入了分布式协同的能力。集群各节点通过使用该引擎和zookeeper协同，来实现集群节点间的数据同步。




ReplicatedMergeTree与MergeTree的逻辑关系

在MergeTree引擎表中，一个数据分区由开始创建到全部完成，会历经两类存储区域。
（1）内存：数据首先会被写入内存缓冲区。
（2）本地磁盘：数据接着会被写入tmp临时目录分区，待全部完成后再将临时目录重命名为正式分区。
ReplicatedMergeTree引擎在上述基础之上增加了ZooKeeper的部分，它会进一步在ZooKeeper内创建一系列的监听节点，并以此实现多个实例之间的通信。在整个通信过程中，ZooKeeper并不会涉及表数据的传输。

作为数据副本的主要实现载体，ReplicatedMergeTree引擎在设计上有以下特点。

依赖ZooKeeper：在执行INSERT和ALTER查询的时候，ReplicatedMergeTree需要借助ZooKeeper的分布式协同能力，以实现多个副本之间的同步。查询副本的时候，不需要使用ZooKeeper。

表级别的副本：副本是在表级别定义的，每张表的副本配置都可以按照它的实际需求进行个性化定义，包括副本的数量，以及副本在集群内的分布位置等。

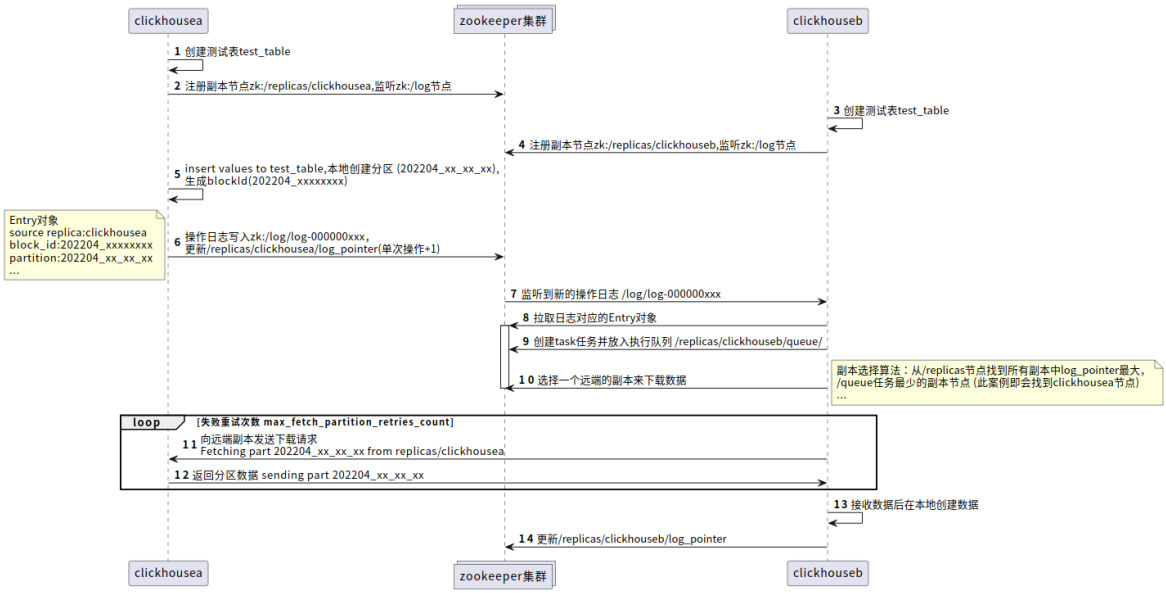
多主架构：可以在任意一个副本上执行INSERT和ALTER查询，它们的效果是相同的。这些操作会借助ZooKeeper的协同能力被分发至每个副本以本地形式执行。

Block数据块：在执行INSERT命令写入数据时，会将数据切分成若干个Block数据块。Block数据块是数据写入的基本单元，并且具有写入的原子性和唯一性。

原子性：在数据写入时，一个Block块内的数据要么全部写入成功，要么全部失败。

唯一性：在写入一个Block数据块的时候，会按照当前Block数据块的数据顺序、数据行和数据大小等指标，计算Hash信息摘要并记录在案。在此之后，如果某个待写入的Block数据块与先前已被写入的Block数据块拥有相同的Hash摘要，则该Block数据块会被忽略。

以clickhousea和clickhouseb的互为副本为例，插入数据同步流程如下：



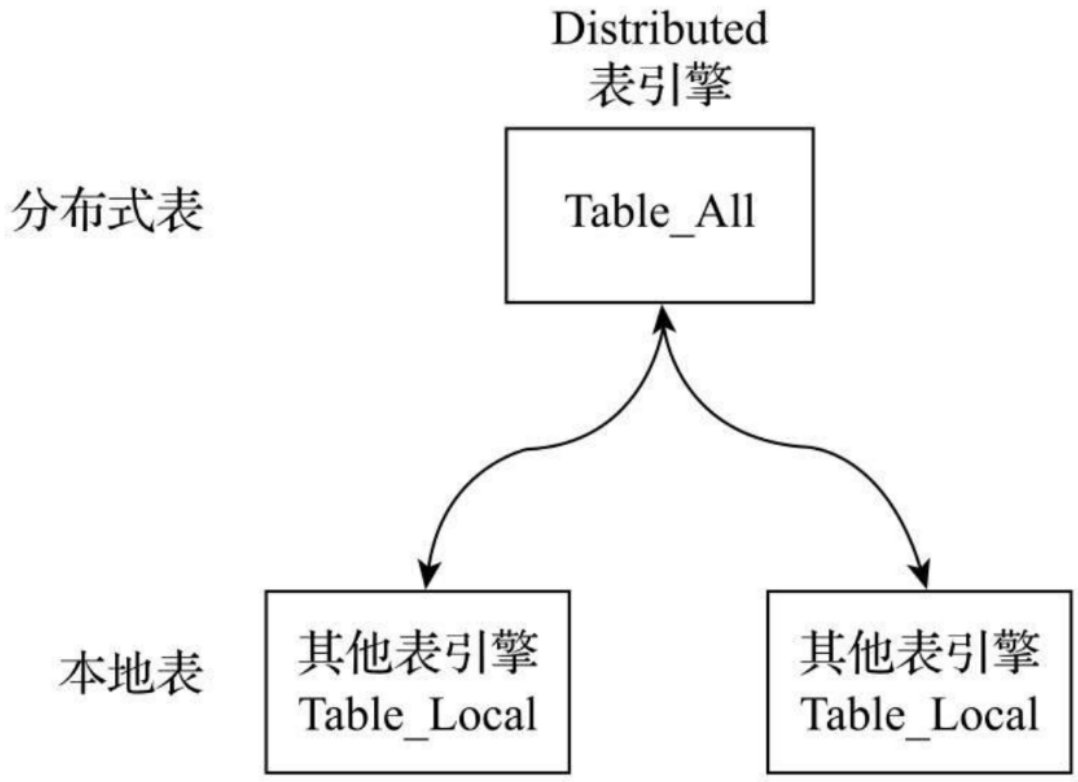

在`INSERT`的写入过程中，ZooKeeper不会进行任何实质性的数据传输。本着谁执行谁负责的原则，在这个案例中由clickhousea首先在本地写入了分区数据。之后，也由这个副本负责发送Log日志，通知其他副本下载数据。

# 4.技术方案二

## 4.1【分片模式】

**分片模式：**在集群的各个节点上，每个节点都拥有不同的数据，使用分片的主要目的是实现数据的水平切分。

ClickHouse集群中的每个服务节点都可称为一个分片。从理论上来讲，假设有N(N >= 1)张数据表A，分布在N个ClickHouse服务节点，而这些数据表彼此之间没有重复数据，那么就可以说数据表A拥有N个分片。然而在工程实践中，如果只有这些分片表，那么整个分片方案基本是不可用的。对于一个完整的方案来说，还需要考虑数据在写入时，如何被均匀地写至各个分片，以及数据在查询时，如何路由到每个shard，并组合成结果集。所以，ClickHouse的数据分片需要结合Distributed表引擎一同使用，如图所示。




Distributed表引擎自身不存储任何数据，它能够作为分布式表的一层透明代理，在集群内部自动开展数据的写入、分发、查询、路由等工作。

采用分片模式的clickhouse集群uml图如下：

clickhousea，clickhouseb，clickhousec这三个节点共同组成clickhouse集群，3个节点保存的数据均不同。

## 4.2 关键技术

通过引入数据副本，虽然能够有效降低数据的丢失风险（多份存储），并提升查询的性能（分摊查询、读写分离），但是仍然有一个问题没有解决，那就是数据表的容量问题。每个副本自身，都保存了数据表的全量数据。所以在业务量十分庞大的场景中，依靠副本并不能解决单表的性能瓶颈。想要从根本上解决这类问题，需要借助另外一种手段，将数据水平切分，即数据分片。

clickhouse 的数据分片需要结合 DIstributed 表引擎一起使用。**DIstributed 表本身不存储任何数据**，它能够作为分布式表的一层代理，在集群内部自动展开数据写入、分发、查询、路由等工作。

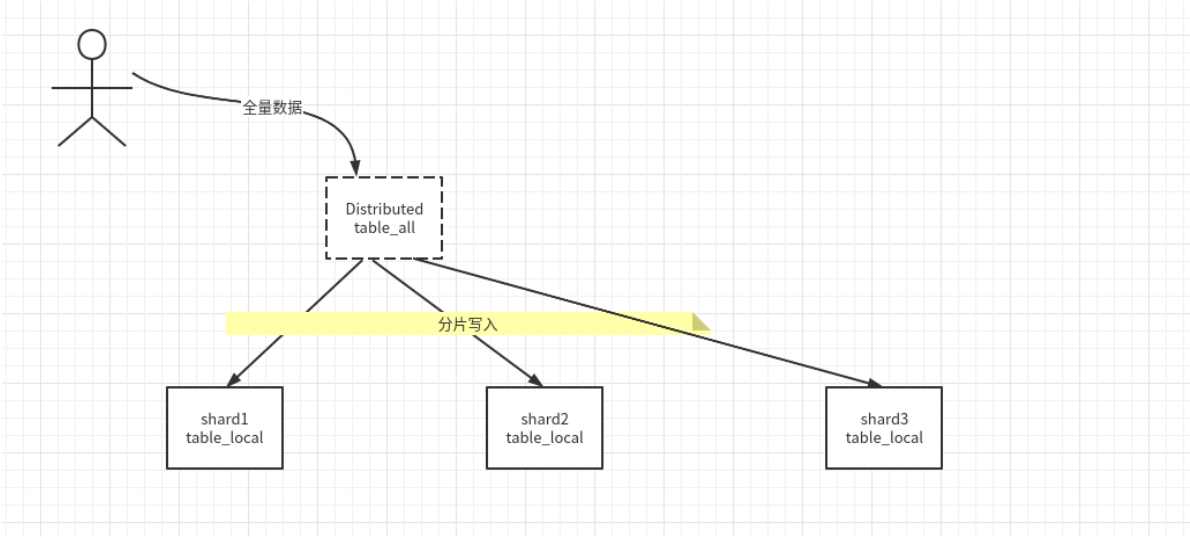
在分片模式的clickhouse节点中通常存在本地表和分布式表：

本地表：通常以*local为后缀进行命名。本地表是承接数据的载体，可以使用非Distributed的任意表引擎，一张本地表对应了一个数据分片。*
*分布式表：通常以*all为后缀进行命名。分布式表只能使用Distributed表引擎，它与本地表形成一对多的映射关系，日后将通过分布式表代理操作多张本地表。

分片写入流程：

在向集群内的分片写入数据时，通常有两种思路

1.借助外部计算系统，事先将数据均匀分片，再借由计算系统直接将数据写入ClickHouse集群的各个本地表。
2.通过Distributed表引擎代理写入分片数据。
第一种方案通常拥有更好的写入性能，因为分片数据是被并行点对点写入的。但是这种方案的实现主要依赖于外部系统，而不在于ClickHouse自身。

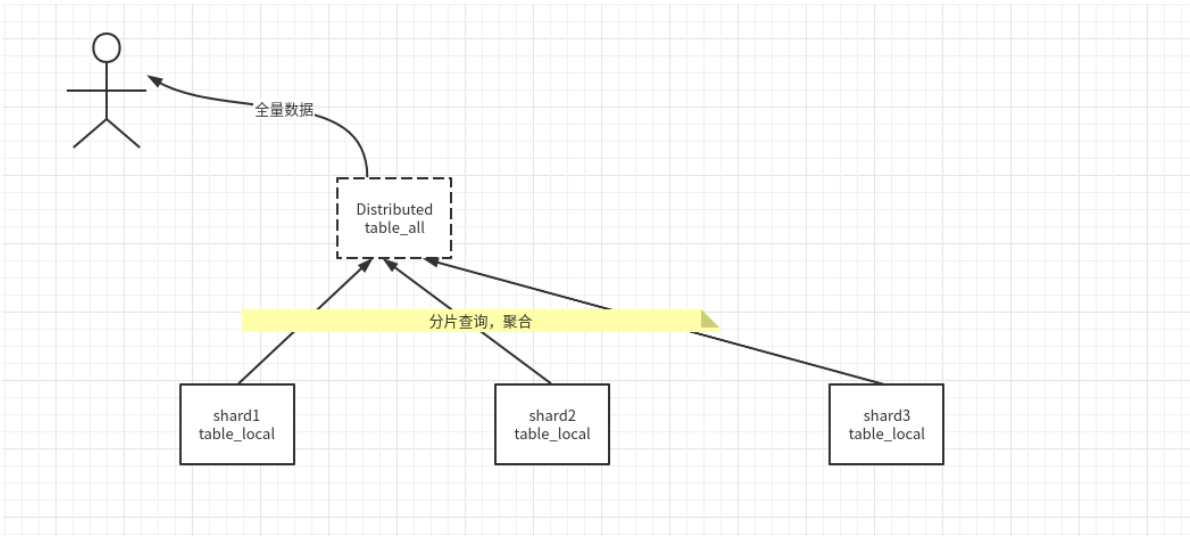



分片查询流程：

与数据写入有所不同，在面向集群查询数据的时候，**只能**通过Distributed表引擎实现。当Distributed表接收到`SELECT`查询的时候，它会依次查询每个分片的数据，再合并汇总返回，流程如下：

1.**查询各个分片数据**：One和Remote步骤是并行执行的，它们分别负责了本地和远端分片的查询动作。

2.**合并返回结果**：多个分片数据均查询返回后，在执行节点将所有数据`union`合并



# 5.技术方案三

## 5.1 【分片&副本综合模式】

分片&副本综合模式，是上述两种方案的综合，首先采用distributed表引擎的分片，然后针对每个分片采用replicated引擎的副本。

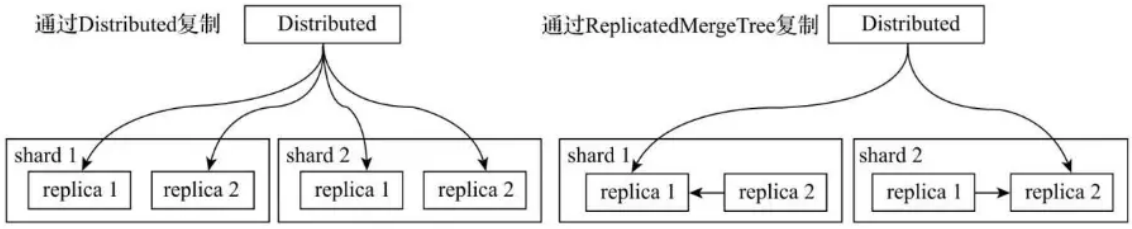
采用分片&副本模式的clickhouse集群uml图如下：

3个shard仍然是通过distributed引擎进行分片关联，虚线方框内是一个分片的集群，例如clickhousea_replica1和clickhousea_replica2互为副本。每个分片都能允许其中一个clickhouse节点宕机，整体服务仍然可用。

## 5.2 关键技术

该方案为上述两种方案的综合应用，分片与副本的原理不再赘述。

还有一点补充，数据在多个副本之间，有两种复制实现方式：一种是继续借助Distributed表引擎，由它将数据写入副本；另一种则是借助ReplicatedMergeTree表引擎实现副本数据的分发。两种方式的区别如图所示：




1.通过Distributed复制数据（internal_replication=false）

在这种实现方式下，即使本地表不使用ReplicatedMergeTree表引擎，也能实现数据副本的功能。Distributed会同时负责分片和副本的数据写入工作，而副本数据的写入流程与分片逻辑相同。

2.通过ReplicatedMergeTree复制数据（internal_replication=true）

该方式需要在集群的shard配置中增加internal_replication参数并将其设置为true（默认为false），在shard内，多个replica副本之间的数据复制会交由ReplicatedMergeTree引擎处理，不再由Distributed负责，从而为其减负。

# 6.实验验证

首先搭建好zookeeper集群作为外部组件备用，本案例使用docker swarm模拟了三台zookeeper的集群，分别为

zoo1 ==> 10.20.23.109:2181;

zoo2 ==> 10.20.23.109:2182;

zoo3 ==> 10.20.23.109:2183;

compose.yml文件可参考如下配置：

```
  zoo1:
    image: zookeeper:3.5.8
    networks:
      - base-gateway
    hostname: zoo1
    ports:
      - 2181:2181
    environment:
      ZOO_MY_ID: 1
      ZOO_SERVERS: server.1=0.0.0.0:2888:3888;2181 server.2=zoo2:2888:3888;2181
server.3=zoo3:2888:3888;2181
  zoo2:
    image: zookeeper:3.5.8
    hostname: zoo2
    networks:
      - base-gateway
    ports:
      - 2182:2181
    environment:
      ZOO_MY_ID: 2
      ZOO_SERVERS: server.1=zoo1:2888:3888;2181 server.2=0.0.0.0:2888:3888;2181
server.3=zoo3:2888:3888;2181
  zoo3:
    image: zookeeper:3.5.8
    networks:
      - base-gateway
    hostname: zoo3
    ports:
      - 2183:2181
    environment:
      ZOO_MY_ID: 3
      ZOO_SERVERS: server.1=zoo1:2888:3888;2181 server.2=zoo2:2888:3888;2181
server.3=0.0.0.0:2888:3888;2181
```

## 6.1 副本模式验证

搭建三个clickhouse服务节点，将各个节点的clickhouse-server.config.xml配置文件中的 配置段注释打开，增加已有的zookeeper集群地址如下：

```
    <zookeeper>
        <node>
            <host>10.20.23.109</host>
            <port>2181</port>
        </node>
        <node>
            <host>10.20.23.109</host>
            <port>2182</port>
        </node>
        <node>
            <host>10.20.23.109</host>
            <port>2183</port>
        </node>
    </zookeeper>
```

分别启动三个clickhouse服务节点，分别为clickhousea，clickhouseb，clickhousec，如下所示：

```
wang@wang-PC:~/Desktop/cluster$ docker service ls |grep clickhouse
w8mlqz34de5c        base_clickhousea    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
zoygmgwlvcyf        base_clickhouseb    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
j9357rtt4a9h        base_clickhousec    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
```

分别进入三个clickhouse服务节点，执行ReplicatedMergeTree引擎建表命令:

```
create table test(id Int16)
engine=ReplicatedMergeTree('/clickhouse/tables/oneshard/test','{replica}')
order by id;

{replica} 代表clickhouse节点的hostname  分别对应 clickhousea,clickhouseb,clickhousec
```

```
clickhousea :) create table test(id Int16)
:-] engine=ReplicatedMergeTree('/clickhouse/tables/oneshard/test','{rep
lica}')
:-] order by id;

CREATE TABLE test
(
    `id` Int16
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/oneshard/test', '{repl
ica}')
ORDER BY id

Query id: c92074d8-1f6d-4d2c-8a65-61af93ae1951

Ok.

0 rows in set. Elapsed: 0.137 sec.

clickhousea :)
```

```
clickhouseb :) create table test(id Int16)
:-] engine=ReplicatedMergeTree('/clickhouse/tables/oneshard/test','{replic
a}')
:-] order by id;

CREATE TABLE test
(
    `id` Int16
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/oneshard/test', '{replica
}')
ORDER BY id

Query id: e6df9692-2907-4626-a0c5-323b681bbe06

Ok.

0 rows in set. Elapsed: 0.138 sec.

clickhouseb :)
```

```
clickhousec :) create table test(id Int16)
:-] engine=ReplicatedMergeTree('/clickhouse/tables/oneshard/test','{rep
lica}')
:-] order by id;

CREATE TABLE test
(
    `id` Int16
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/oneshard/test', '{repl
ica}')
ORDER BY id

Query id: 1ddf2dc9-6f49-4442-9d2d-8a913ca07ca1

Ok.

0 rows in set. Elapsed: 0.144 sec.

clickhousec :)
```

接下来进行数据插入的模拟，随机选择任意一个节点clickhouseb执行如下命令

```
clickhouseb :) insert into test values(222)

INSERT INTO test VALUES

Query id: 3e8a4f6f-2285-4569-86a0-f2dff3be9a83

Ok.

1 rows in set. Elapsed: 0.040 sec.

clickhouseb :)
```

然后在剩余两个节点查看表的数据：

```
clickhousea :) select * from test

SELECT *
FROM test

Query id: d87e2b8e-4817-45fe-8ee6-4890e243018b

┌─id─┐
│ 222 │
└─────┘

1 rows in set. Elapsed: 0.002 sec.

clickhousea :)
```

```
clickhousec :) select * from test

SELECT *
FROM test

Query id: 1717ae68-5a39-4880-9f8d-1c5457977847

┌─id─┐
│ 222 │
└─────┘

1 rows in set. Elapsed: 0.002 sec.

clickhousec :)
```

由此可以验证，数据已在三个节点中完成了同步复制。

同样的在clickhousea和clickhouseb写入的其他数据也会同步其他两个节点，过程不再截图。

**任意节点写入的数据均会同步到集群中的所有其他节点**。

### 6.1.1 节点宕机模拟

随机选取一个节点clickhousec进行宕机模拟，关闭该服务节点模拟宕机情景:

```
wang@wang-PC:~/Desktop/replicated$ docker service ls |grep clickhouse
290cap7ytr9x        base_clickhousea    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
48xy7vlto03r        base_clickhouseb    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
yh8jv1ry1v58        base_clickhousec    replicated          0/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
wang@wang-PC:~/Desktop/replicated$
```

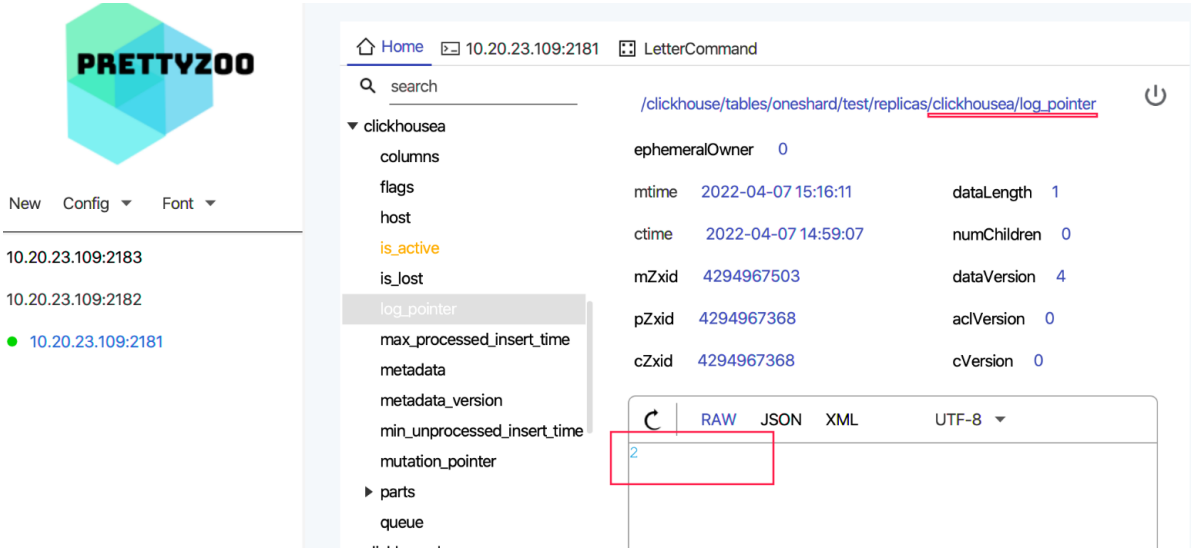
选择剩余两个正常节点中的clickhousea写入数据，能够马上同步到clickhouseb。

```
clickhousea :) insert into test values(136);

INSERT INTO test VALUES

Query id: 7bb53c74-958d-452a-b2f2-22ff5441d416

Ok.

1 rows in set. Elapsed: 0.027 sec.

clickhousea :) select * from test

SELECT *
FROM test

Query id: 869a0b8d-0ace-46da-a0e2-ecdc07bea00f

┌──id─┐
│ 222 │
└─────┘
┌──id─┐
│ 136 │
└─────┘
```

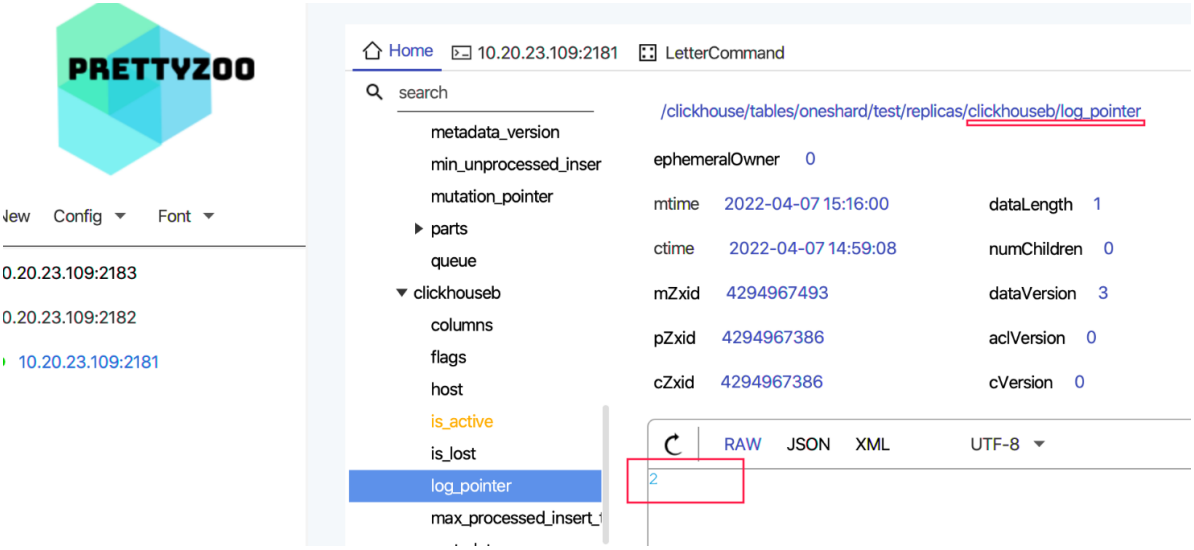
```
clickhouseb :) select * from test

SELECT *
FROM test

Query id: 12e5e9f0-3ba8-4048-9a31-a9c0d59ed5bf

┌──id─┐
│ 136 │
└─────┘
┌──id─┐
│ 222 │
└─────┘

2 rows in set. Elapsed: 0.004 sec.

clickhouseb :)
```

观察zookeeper的kv，可以看到clickhousea 和clickhouseb 的log_pointer均为2（因为这是第二个插入操作），**而clickhousec的log_pointer仍然为1**

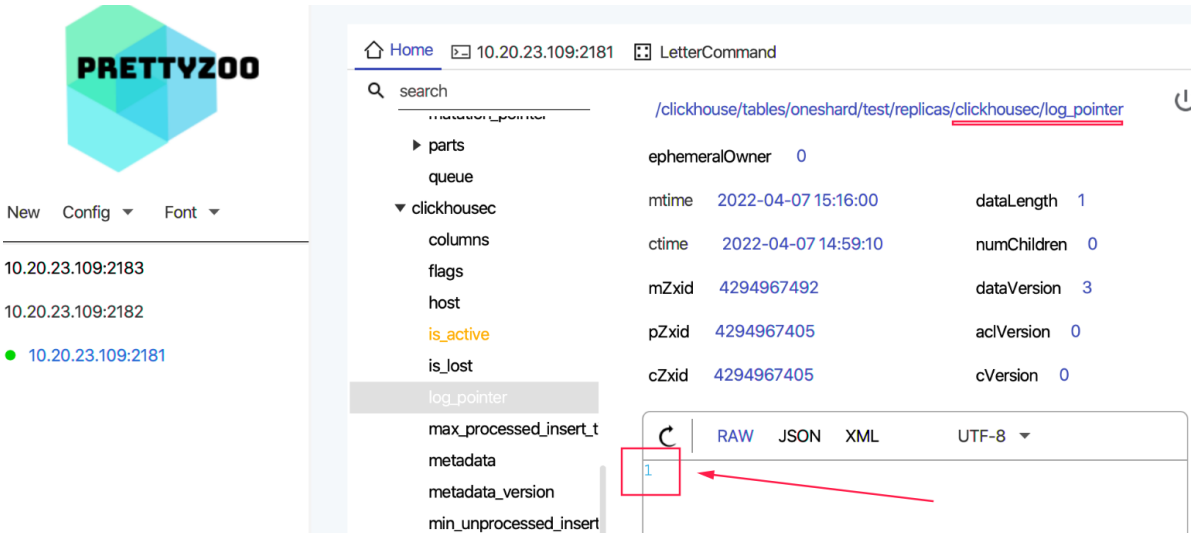
**clickhousea:**




**clickhouseb:**




**clickhousec:**




这说明clickhousec只同步到了第一步操作的数据。

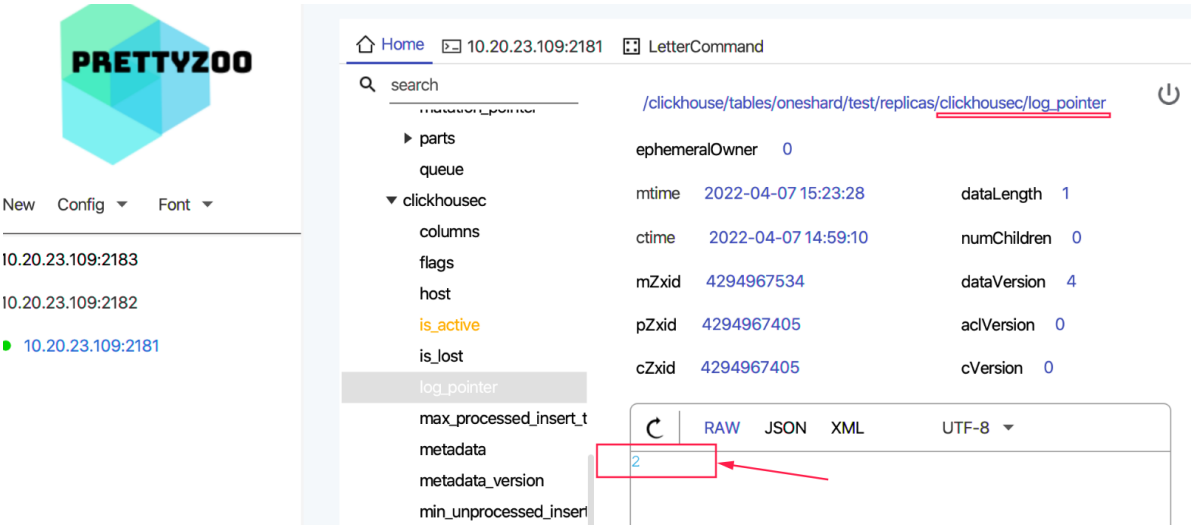
接下来启动clickhousec，等待其启动完成

```
wang@wang-PC:~/Desktop/cluster$ docker service ls |grep clickhouse
w8mlqz34de5c        base_clickhousea    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
zoygmgwlvcyf        base_clickhouseb    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
j9357rtt4a9h        base_clickhousec    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
```

再进入clickhousec中查询数据

```
clickhousec :) select * from test

SELECT *
FROM test

Query id: f96ab8d5-af94-4910-91c5-a91223510087

┌──id─┐
│ 136 │
└─────┘
┌──id─┐
│ 222 │
└─────┘

2 rows in set. Elapsed: 0.003 sec.

clickhousec :)
```

可以看到数据也同步完成了，此刻我们检查zookeeper中的kv可以**发现clickhousec的log_pointer也变为了2**




**说明其故障宕机恢复后会自动完成数据同步，其正是通过replicatedMergetree引擎集成zookeeper来实现同步机制。**

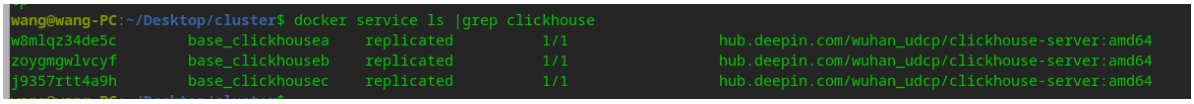
## 6.2 分片模式验证

搭建三个clickhouse服务节点，将各个节点的clickhouse-server.config.xml配置文件中的 <remote_servers> 配置段注释打开，并分别将clickhouse节点配置填入shard配置段中：

```
    <remote_servers>
        <testcluster>
            <shard>
                <replica>
                    <host>clickhousea</host>
```

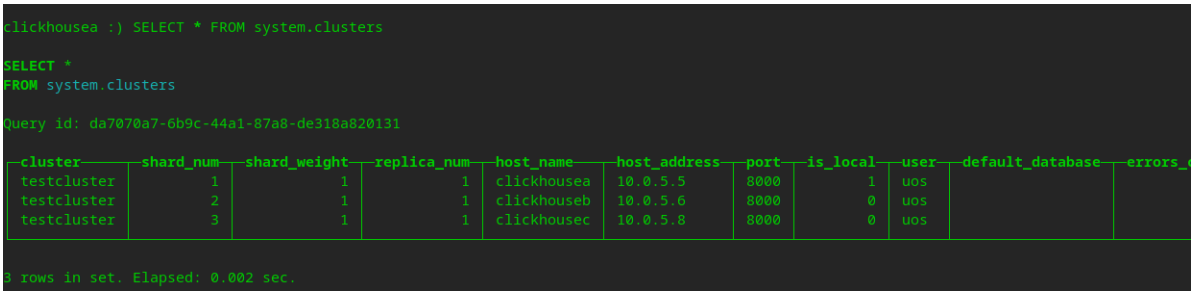
```
                    <port>8000</port>
                    <user>uos</user>
                    <password>Udcp2022cs</password>
                </replica>
            </shard>
            <shard>
                <replica>
                    <host>clickhouseb</host>
                    <port>8000</port>
                    <user>uos</user>
                    <password>Udcp2022cs</password>
                </replica>
            </shard>
            <shard>
                <replica>
                    <host>clickhousec</host>
                    <port>8000</port>
                    <user>uos</user>
                    <password>Udcp2022cs</password>
                </replica>
            </shard>
        </testcluster>
    </remote_servers>
```

分别启动三个clickhouse服务节点，分别为clickhousea，clickhouseb，clickhousec，如下所示：

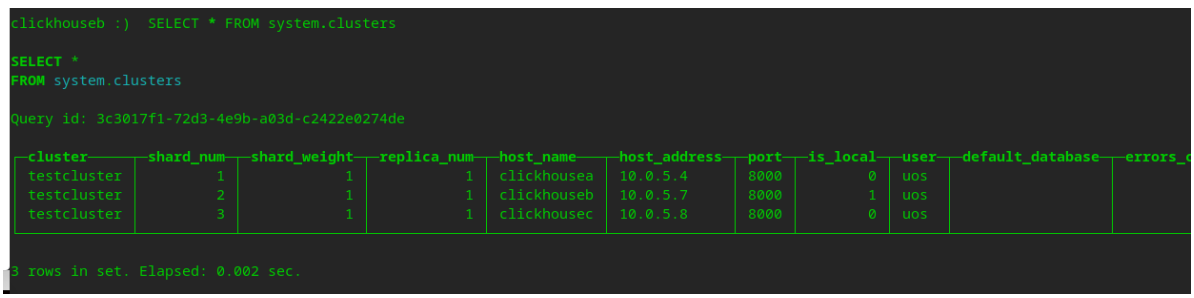



三个节点均能看到如下的cluster集群信息，可以看到shard_num代表分片序号，is_local代表本节点等等

clickhousea:




clickhouseb:




clickhousec:

```
clickhousec :) SELECT * FROM system.clusters

SELECT *
FROM system.clusters

Query id: 32855e62-bf73-4b7f-bd4c-a2d97643bfef

┌─cluster─────┬─shard_num─┬─shard_weight─┬─replica_num─┬─host_name──┬─host_address─┬─port─┬─is_local─┬─user─┬─default_database─┬─errors_
│ testcluster │         1 │            1 │           1 │ clickhousea │ 10.0.5.4     │ 8000 │        0 │ uos  │                  │
│ testcluster │         2 │            1 │           1 │ clickhouseb │ 10.0.5.6     │ 8000 │        0 │ uos  │                  │
│ testcluster │         3 │            1 │           1 │ clickhousec │ 10.0.5.9     │ 8000 │        1 │ uos  │                  │
└─────────────┴───────────┴──────────────┴─────────────┴─────────────┴──────────────┴──────┴──────────┴──────┴──────────────────┴─────────

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

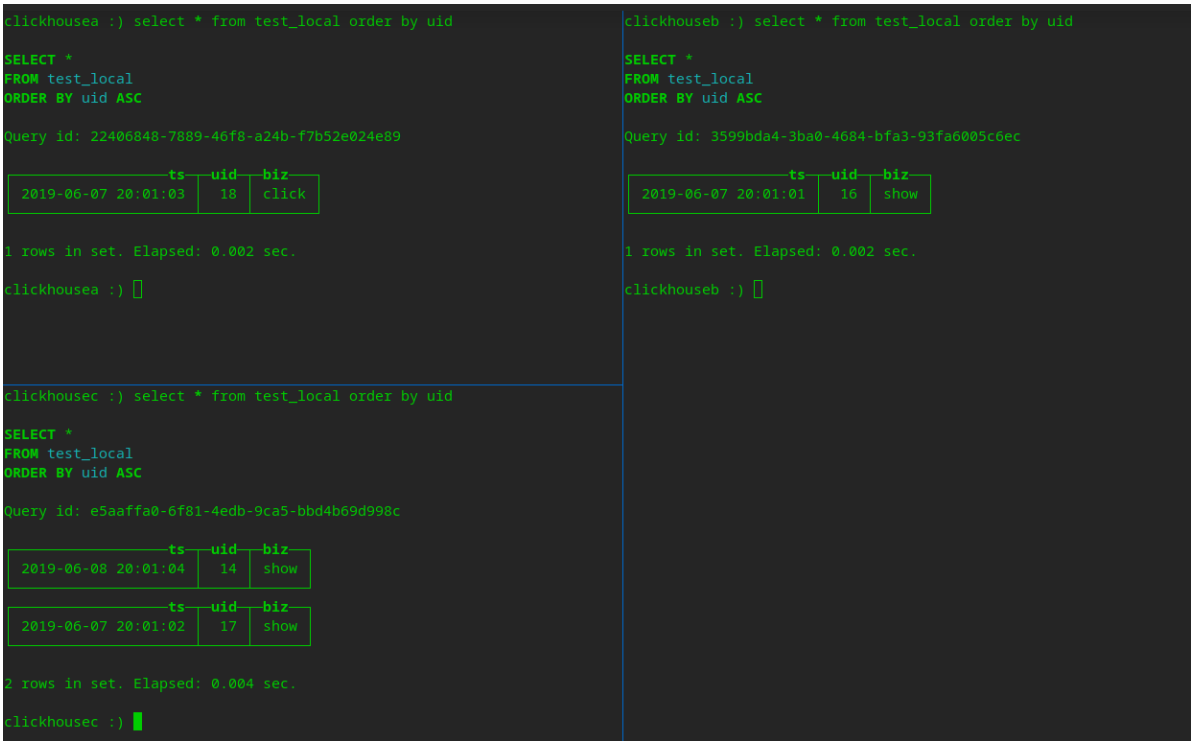
分别进入三个clickhouse服务节点，执行如下的建本地表和分布式表命令:

```
#本地表
CREATE TABLE test_local
(
    `ts` DateTime,
    `uid` Int64,
    `biz` String
)
ENGINE = MergeTree
PARTITION BY toYYYYMMDD(ts)
ORDER BY ts
SETTINGS index_granularity = 8192;

#分布式表
create table test_all  (
  `ts` DateTime,
  `uid` Int64,
  `biz` String
) engine=Distributed(testcluster, default, test_local, uid);

#Distributed引擎关键字跟的内容分别是：集群名，数据库名，表名，分片健
```

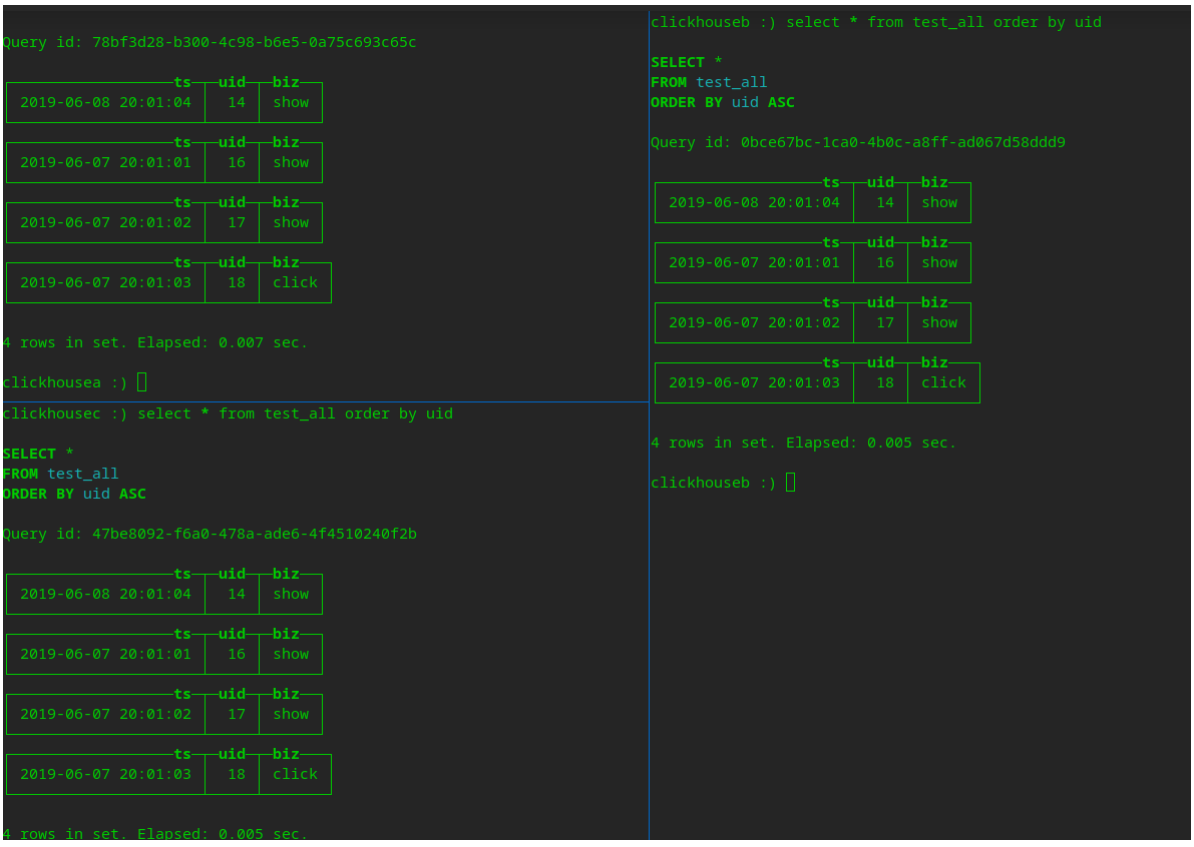
表建好后，随机选择一个节点 clickhousec 进行插入数据操作，写入分布式表 test_all:

```
clickhousec :) INSERT INTO test_all VALUES ('2019-06-07 20:01:01', 16,
'show'),
:-]                         ('2019-06-07 20:01:02', 17, 'show'),
:-]                         ('2019-06-07 20:01:03', 18, 'click'),
:-]                         ('2019-06-08 20:01:04',14, 'show');

INSERT INTO test_all VALUES

Query id: 8ab9d75e-a664-4949-9182-866022860dc5

Ok.

4 rows in set. Elapsed: 0.012 sec.

clickhousec :)
```

观察数据是否写到不同的分片节点:




同样的，在任意节点读取分布式表都能获取到全量的数据：




**DIstributed 表本身不存储任何数据，它能够作为分布式表的一层代理，在集群内部自动展开数据写入、分发、查询、路由等工作。**

### 6.2.1 节点宕机模拟

随机选取一个节点clickhousec进行宕机模拟，关闭该服务节点模拟宕机情景:

```
wang@wang-PC:~/Desktop/cluster$ docker service ls |grep click
kmy5myevobhs        base_clickhousea    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
xq1a0m0b429d        base_clickhouseb    replicated          0/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
duzmxkovop8q        base_clickhousec    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
wang@wang-PC:~/Desktop/cluster$
```

进入剩余两个节点，执行查询操作:

```
clickhousea :) select * from test_local order by uid

SELECT *
FROM test_local
ORDER BY uid ASC

Query id: b27763cd-7c11-4d0e-8222-521fbb86e536

┌──────────────────ts─┬─uid─┬─biz─┐
│ 2019-06-07 20:01:03 │  18 │ click │
└─────────────────────┴─────┴─────┘

1 rows in set. Elapsed: 0.004 sec.

clickhousea :)
```

```
clickhousec :) select * from test_local order by uid

SELECT *
FROM test_local
ORDER BY uid ASC

Query id: a6db379f-1b68-412e-991b-8fa3c360fe1f

┌──────────────────ts─┬─uid─┬─biz──┐
│ 2019-06-08 20:01:04 │  14 │ show │
└─────────────────────┴─────┴──────┘
┌──────────────────ts─┬─uid─┬─biz──┐
│ 2019-06-07 20:01:02 │  17 │ show │
└─────────────────────┴─────┴──────┘

2 rows in set. Elapsed: 0.004 sec.

clickhousec :)
```

**查询本地表能够获得正常响应。**

```
clickhousea :) select * from test_all order by uid

SELECT *
FROM test_all
ORDER BY uid ASC

Query id: 1b4a932e-11cf-4c6c-a6c9-407e490e5953

↑ Progress: 1.00 rows, 26.00 B (9.65 rows/s., 250.79 B/s.)  99%
0 rows in set. Elapsed: 0.114 sec.

Received exception from server (version 21.7.2):
Code: 279. DB::Exception: Received from localhost:8000. DB::Exception:
All connection tries failed. Log:

Code: 32, e.displayText() = DB::Exception: Attempt to read after eof (v
ersion 21.7.2.7 (official build))
Code: 209, e.displayText() = DB::NetException: Timeout: connect timed o
ut: 10.0.6.4:8000 (clickhouseb:8000) (version 21.7.2.7 (official build)
)
Code: 209, e.displayText() = DB::NetException: Timeout: connect timed o
ut: 10.0.6.4:8000 (clickhouseb:8000) (version 21.7.2.7 (official build)
)

Query id: 47be8092-f6a0-478a-ade6-4f4510240f2b
```

```
clickhousec :) select * from test_all order by uid

SELECT *
FROM test_all
ORDER BY uid ASC

Query id: 132fa73b-d182-42ee-a887-6eddd4e24225

↖ Progress: 3.00 rows, 76.00 B (28.85 rows/s., 730.75 B/s.)  99%
0 rows in set. Elapsed: 0.108 sec.

Received exception from server (version 21.7.2):
Code: 279. DB::Exception: Received from localhost:8000. DB::Exception: Al
 connection tries failed. Log:

Code: 32, e.displayText() = DB::Exception: Attempt to read after eof (ver
ion 21.7.2.7 (official build))
Code: 209, e.displayText() = DB::NetException: Timeout: connect timed out
 10.0.6.4:8000 (clickhouseb:8000) (version 21.7.2.7 (official build))
Code: 209, e.displayText() = DB::NetException: Timeout: connect timed out
 10.0.6.4:8000 (clickhouseb:8000) (version 21.7.2.7 (official build))

: While executing Remote.

clickhousec :)
```

**查询分布式表均会报timeout，无法连接clickhouseb的错误，无法获得完整的数据。**

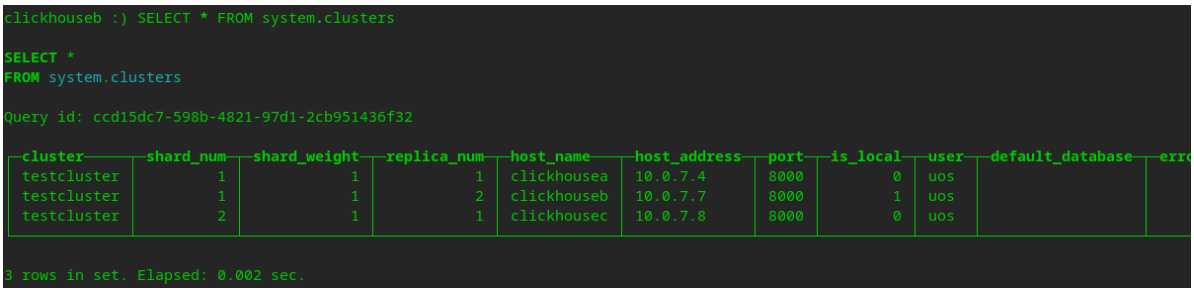
## 6.3 分片&副本模式验证

搭建三个clickhouse服务节点，将各个节点的clickhouse-server.config.xml配置文件中的<remote_server> 和 配置段注释打开，并配置如下:

```
    <remote_servers>
       <testcluster>
           <shard>
               <internal_replication>true</internal_replication>
               <replica>
                   <host>clickhousea</host>
                   <port>8000</port>
                   <user>uos</user>
                   <password>Udcp2022cs</password>
```

```
                </replica>
                <replica>
                    <host>clickhouseb</host>
                    <port>8000</port>
                    <user>uos</user>
                    <password>Udcp2022cs</password>
                </replica>
            </shard>
            <shard>
                <replica>
                    <host>clickhousec</host>
                    <port>8000</port>
                    <user>uos</user>
                    <password>Udcp2022cs</password>
                </replica>
            </shard>
        </testcluster>
    </remote_servers>



    <zookeeper>
        <node>
            <host>10.20.23.109</host>
            <port>2181</port>
        </node>
        <node>
            <host>10.20.23.109</host>
            <port>2182</port>
        </node>
        <node>
            <host>10.20.23.109</host>
            <port>2183</port>
        </node>
    </zookeeper>
```

**该配置代表使用3个节点搭建集群，分片数量为2，分片1中clickhousea和clickhouseb互为副本，分片2为单独的clickhousec节点。如下图示:**



```
clickhouseb :) SELECT * FROM system.clusters

SELECT *
FROM system.clusters

Query id: ccd15dc7-598b-4821-97d1-2cb951436f32

┌─cluster─────┬─shard_num─┬─shard_weight─┬─replica_num─┬─host_name──┬─host_address─┬─port─┬─is_local─┬─user─┬─default_database─┬─erro
│ testcluster │         1 │            1 │           1 │ clickhousea │ 10.0.7.4     │ 8000 │        0 │ uos  │                  │
│ testcluster │         1 │            1 │           2 │ clickhouseb │ 10.0.7.7     │ 8000 │        1 │ uos  │                  │
│ testcluster │         2 │            1 │           1 │ clickhousec │ 10.0.7.8     │ 8000 │        0 │ uos  │                  │
└─────────────┴───────────┴──────────────┴─────────────┴─────────────┴──────────────┴──────┴──────────┴──────┴──────────────────┴─────

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

进入任意一个节点，进行建表操作，使用on cluster 关键字可以在集群环境下自动在每个节点都创建好本地表和分布式表:

```
CREATE TABLE test_local ON CLUSTER testcluster
(
    `ts` DateTime,
    `uid` Int64,
    `biz` String
)
```

```
ENGINE = ReplicatedMergeTree('/clickhouse/cluster/tables/{shard}/test_local',
'{replica}')
PARTITION BY toYYYYMMDD(ts)
ORDER BY ts
SETTINGS index_granularity = 8192;



create table test_all on CLUSTER testcluster (
      `ts` DateTime,
      `uid` Int64,
      `biz` String
) engine=Distributed(testcluster, default, test_local, uid);
```

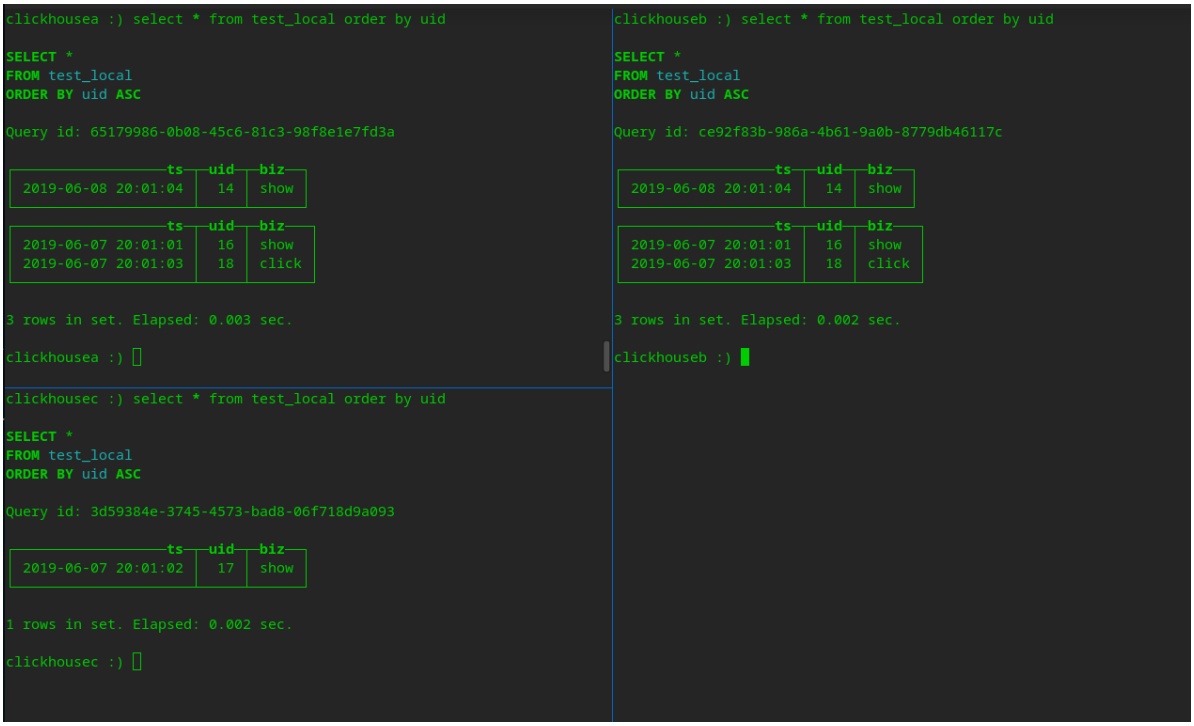
```
CREATE TABLE test_local ON CLUSTER testcluster
(
    `ts` DateTime,
    `uid` Int64,
    `biz` String
)
ENGINE = ReplicatedMergeTree('/clickhouse/cluster/tables/{shard}/test_local', '{replica}')
PARTITION BY toYYYYMMDD(ts)
ORDER BY ts
SETTINGS index_granularity = 8192

Query id: 7f64ab5c-a436-4cb7-bafe-bcbb92b96afc

┌─host──────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
│ clickhousec │ 8000 │      0 │       │                   2 │                1 │
│ clickhouseb │ 8000 │      0 │       │                   1 │                1 │
└─────────┴────┴──────┴─────┴─────────────┴──────────┘
┌─host──────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
│ clickhousea │ 8000 │      0 │       │                   0 │                0 │
└─────────┴────┴──────┴─────┴─────────────┴──────────┘

3 rows in set. Elapsed: 0.361 sec.

clickhousec :)
```

```
CREATE TABLE test_all ON CLUSTER testcluster
(
    `ts` DateTime,
    `uid` Int64,
    `biz` String
)
ENGINE = Distributed(testcluster, default, test_local, uid)

Query id: ab77808b-8f31-4939-aee9-839bf6f8a2f9

┌─host──────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
│ clickhousec │ 8000 │      0 │       │                   2 │                0 │
│ clickhousea │ 8000 │      0 │       │                   1 │                0 │
│ clickhouseb │ 8000 │      0 │       │                   0 │                0 │
└─────────┴────┴──────┴─────┴─────────────┴──────────┘

3 rows in set. Elapsed: 0.141 sec.

clickhousec :)
```

表建好后，前往任意节点通过分布式表模拟插入4条实验数据

```
clickhousec :)
clickhousec :) INSERT INTO test_all VALUES ('2019-06-07 20:01:01', 16, 'show'),
:-]                        ('2019-06-07 20:01:02', 17, 'show'),
:-]                        ('2019-06-07 20:01:03', 18, 'click'),
:-]                        ('2019-06-08 20:01:04',14, 'show');

INSERT INTO test_all VALUES

Query id: c4775c2e-cce4-42a1-943f-60008d7e4991

Ok.

4 rows in set. Elapsed: 0.048 sec.

clickhousec :)
```

在每个节点上查询本地表 test_local




**可以看到插入的4条数据被分片到了两块，clickhousea和clickhouseb作为分片1被分配了3条数据，同时clickhousea和clickhouseb之间互为备份，clickhousec作为分片2被分配了1条数据。**

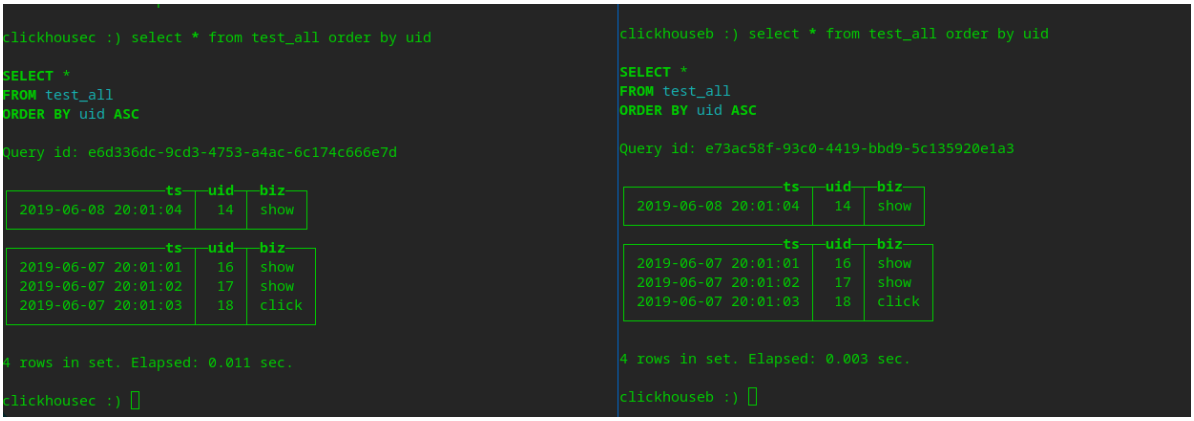
### 6.3.1 节点宕机模拟

由于clickhousea和clickhouseb共同作为了分片1的副本，因此整个clickhouse集群服务允许clickhousea和clickhouseb中的任意一台宕机，不会影响到数据的读写。

选取节点clickhousea进行宕机模拟，关闭该服务节点模拟宕机情景:

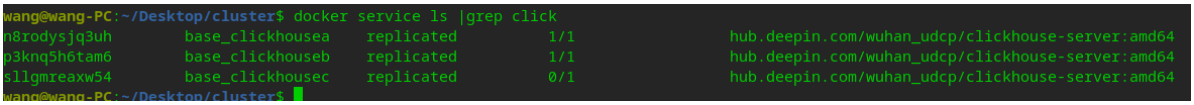
```
wang@wang-PC:~/Desktop/cluster$ docker service ls |grep click
n8rodysjq3uh        base_clickhousea    replicated          0/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
p3knq5h6tam6        base_clickhouseb    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
s1lgmreaxw54        base_clickhousec    replicated          1/1                 hub.deepin.com/wuhan_udcp/clickhouse-server:amd64
wang@wang-PC:~/Desktop/cluster$
```

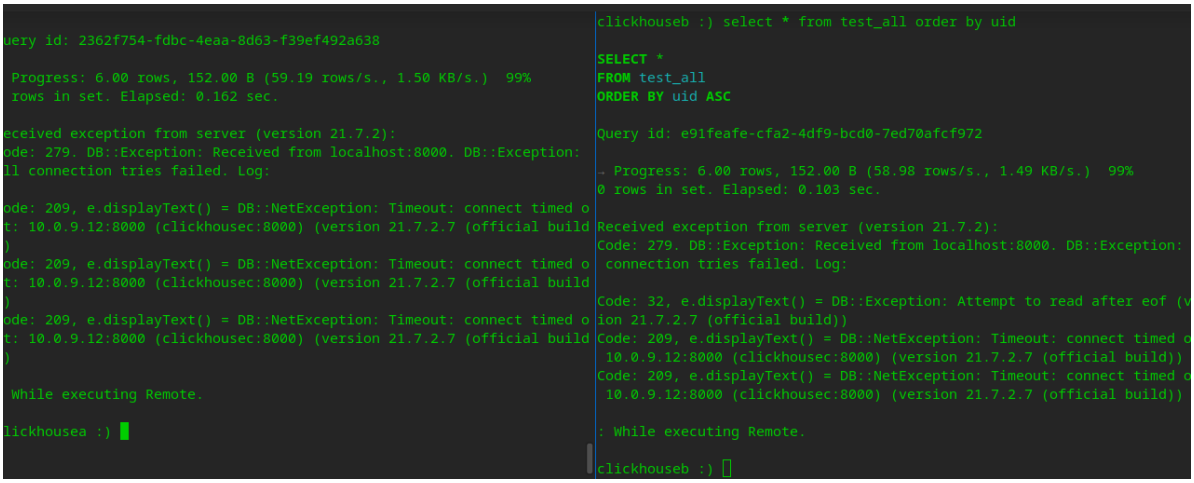
在剩余两个节点中进行分布式表的读取，均能成功获得响应:




**由于clickhousea和clickhouseb互为副本关系，此时往集群中写入数据也能成功，待clickhousea恢复服务后，将会通过zookeeper进行数据的同步，参考方案一，过程不再赘述。**

**同样的，由于clickhousec无副本的原因，因此本案例下若clickhousec宕机，参考方案二，剩余两个节点的服务读取也会报错：**




**如果为分片2也增加一个互为副本的节点clickhoused，那么clickhousec和clickhoused也可允许其中一个节点宕机，整体集群仍可保持可用状态，以此类推。**

# 7.方案对比

本报告归结了clickhouse集群的三种技术方案，实际涉及到的原理主要还是副本与分片两种，方案三为两种方式的综合。

当前没有量化的性能比较数据。简单从功能，安全，兼容、易用性上做对比。

方案一

依靠replicatedMergetree引擎和集成zookeeper对clickhouse的指定的表进行全量的数据同步，能够实现集群方案的高可用性，宕机也能自动恢复，每个节点都会保存全量的数据，多主架构，可以通过proxy负载均衡读写，也可以将某一节点当做主要节点，其余节点作为备用节点，当主节点不可用时切换到备用节点。

方案二

分片主要解决了数据横向扩容的痛点，当使用纯分片模式，存在下列两种情况

1.internal_replication=false时

往分布式表里面写入数据时，数据会写入同属一个shard内的所有备份中，但是这个时候是不校验数据一致性的，有可能出现两个备份数据略微不一致的情况，（例如写入clickhousea_replica1成功，**写入clickhousea_replica1的备份clickhousea_replica2的时候有一些没有成功，结果这两个互为备份的表就不一致了**）虽然这种可能性很小。这种被官方称为poor man's replication，从项目的数据安全角度来说不可接受。

2.internal_replication=true时

往分布式表写入数据时，同一个shard内部只有一个表写入了数据，其他的表均不会写入数据，除非有一个宕机另外的作为补充下入，但此时表之间的数据就不同了，同样是不可接受的。

以上是没有集成zookeeper的两种情况，对于存在数据安全的硬伤是无法接受的。

所以不推荐方案二。

方案三

作为方案一和方案二的结合，分片解决了数据横向扩容的痛点，副本保证了数据的安全，但是由于clickhouse内部不支持集群节点之间相互拥有副本，因此实现起来需要的可用节点更多，例如进行3分片，每个分片2个副本的情况，就需要6个clickhouse节点（以此类推），部署复杂性以及运维成本会大量增加。

# 8.小结

- clickhouse集群方案，无论是副本模式还是分片模式，均主要通过clickhouse-server.config.xml的配置文件进行相关配置，已经深入耦合在clickhouse的引擎设计中。
- clickhouse集群数据安全的核心就是副本同步机制，主要借助zookeeper以及replicated表引擎来协作实现。
- 从本报告中可以看出clickhouse对于zookeeper的强依赖性，每次插入数据都会和zookeeper做交互，在高并发的情况下，压力会更大，因此zookeeper作为高可用的分布式服务组件，其部署服务器性能也应该尽可能高，提高zookeeper的性能，不因为zookeeper性能而影响到Clickhouse集群。

结合集中域管平台项目现状与未来期望，从功能，安全，兼容、易用性等方面考虑，笔者更倾向于选择方案一。

# 9.参考资料


[clickhouse official doc]

[【腾讯看点】ClickHouse最优实践与原理剖析]

[github clickhouse]

[tencent cloud]

[ClickHouse Distribute 引擎深度解读]

[github clickhouse-backup]