Clickhouse集群应用、分片、复制

https://www.jianshu.com/p/20639fdfdc99

简介

通常生产环境我们会用集群代替单机，主要是解决两个问题：

- 效率
- 稳定

如何提升效率？一个大大大任务，让一个人干需要一年，拆解一下让12个人同时干，可能只需要1个月。对于数据库来说，就是数据分片。

如何提升稳定性？所谓稳定就是要保证服务时刻都能用，也常说高可用。这就像团队里必须有二把手，老大有事不在，老二要能顶上。对于数据库来说，就是数据备份。

而集群是解决这两个问题的最佳手段。话说，三个臭皮匠，赛过诸葛亮，这就是团队的力量。

几乎所有大数据相关的产品，基本都是以这两个问题为出发点，Clickhouse也不例外。

不同的是，Hadoop系列的集群是服务级别的，而Clickhouse的集群是表上的。例如，一个hdfs集群，所有文件都会切片、备份；而clickhouse集群中，建表时也可以自己决定用不用。习惯了其他大数据产品的人，刚转到clickhouse会感觉这设计太反人类，后文会详细介绍。

安装

我们使用三台机器演示，三个机器分别安装clickhouse

教程：https://www.jianshu.com/p/5f5ee0904bba

数据

实验使用到官方提供的OnTime数据集，先下载下来，并按照文档建表。

教程： https://clickhouse.yandex/docs/en/single/?query=internal_replication#ontime

数据分片

这里再说明一下，分片是为了提高效率。

分片，就像是把鸡蛋放到多个篮子里，降低整体风险，结果可能是部分数据不可用，虽然一定程度上起到了「高可用」的作用，但分片的目的是为了提速。况且，比较严格的场景下，部分不可用也是不可用。

clickhouse需要自己动手定义分片。

```
vim /etc/clickhouse-server/config.xml
```

编辑config.xml文件，搜索remote_servers：

```
<remote_servers incl="clickhouse_remote_servers" >
    ...
</remote_servers>
```

说明：remote_servers就是集群配置，可以直接在此处配置，也可以提出来配置到扩展文件中。incl属性表示可从外部文件中获取节点名为clickhouse_remote_servers的配置内容。

我们使用扩展文件，首先，添加外部扩展配置文件：

```
<include_from>/etc/clickhouse-server/metrika.xml</include_from>
```

然后，编辑配置文件：

```
vim /etc/clickhouse-server/metrika.xml
```

添加内容：

```
<yandex>
<!-- 集群配置 -->
<clickhouse_remote_servers>
    <!-- 3分片1备份 -->
    <cluster_3shards_1replicas>
        <!-- 数据分片1  -->
        <shard>
            <replica>
```

```
            <host>hadoop1</host>
            <port>9000</port>
        </replica>
    </shard>
    <!-- 数据分片2  -->
    <shard>
        <replica>
            <host>hadoop2</host>
            <port> 9000</port>
        </replica>
    </shard>
    <!-- 数据分片3  -->
    <shard>
        <replica>
            <host>hadoop3</host>
            <port>9000</port>
        </replica>
    </shard>
  </cluster_3shards_1replicas>
</clickhouse_remote_servers>
</yandex>
```

说明：

- clickhouse_remote_servers与config.xml中的incl属性值对应；
- cluster_3shards_1replicas是集群名，可以随便取名；
- 共设置3个分片，每个分片只有1个副本；

在其他两台机器上同样操作。

打开clickhouse-client，查看集群：

```
hadoop1 :) select * from system.clusters;

SELECT *
FROM system.clusters

┌─cluster─────────────────────────────────┬─shard_num─┬─shard_weight─┬─replica_num─┬─host_name─┬─host

| cluster_3shards_1replicas |         1 |            1 |           1 | hadoop1   | 192.168.0.6   | 9000 |        1
| default |                  |
| cluster_3shards_1replicas |         2 |            1 |           1 | hadoop2   | 192.168.0.16  | 9000 |        0
| default |                  |
| cluster_3shards_1replicas |         3 |            1 |           1 | hadoop3   | 192.168.0.11  | 9000 |        0
| default |                  |
└─────────────────────────────────────────┴───────────┴──────────────┴─────────────┴───────────┴──────────────┴──────┴─────────


3 rows in set. Elapsed: 0.003 sec.
```

可以看到cluster_3shards_1replicas就是我们定义的集群名称，一共有三个分片，每个分片有一份数据。

建数据表

在三个节点上分别建表：ontime_local

```
CREATE TABLE `ontime_local` (
  `Year` UInt16,
  `Quarter` UInt8,
  `Month` UInt8,
  `DayofMonth` UInt8,
```

```
`DayOfWeek` UInt8,
`FlightDate` Date,
`UniqueCarrier` FixedString(7),
`AirlineID` Int32,
`Carrier` FixedString(2),
`TailNum` String,
`FlightNum` String,
`OriginAirportID` Int32,
`OriginAirportSeqID` Int32,
`OriginCityMarketID` Int32,
`Origin` FixedString(5),
`OriginCityName` String,
`OriginState` FixedString(2),
`OriginStateFips` String,
`OriginStateName` String,
`OriginWac` Int32,
`DestAirportID` Int32,
`DestAirportSeqID` Int32,
`DestCityMarketID` Int32,
`Dest` FixedString(5),
`DestCityName` String,
`DestState` FixedString(2),
`DestStateFips` String,
`DestStateName` String,
`DestWac` Int32,
`CRSDepTime` Int32,
`DepTime` Int32,
`DepDelay` Int32,
`DepDelayMinutes` Int32,
`DepDel15` Int32,
`DepartureDelayGroups` String,
`DepTimeBlk` String,
`TaxiOut` Int32,
`WheelsOff` Int32,
`WheelsOn` Int32,
`TaxiIn` Int32,
`CRSArrTime` Int32,
`ArrTime` Int32,
`ArrDelay` Int32,
`ArrDelayMinutes` Int32,
`ArrDel15` Int32,
`ArrivalDelayGroups` Int32,
`ArrTimeBlk` String,
`Cancelled` UInt8,
`CancellationCode` FixedString(1),
`Diverted` UInt8,
`CRSElapsedTime` Int32,
`ActualElapsedTime` Int32,
`AirTime` Int32,
`Flights` Int32,
`Distance` Int32,
`DistanceGroup` UInt8,
```

`CarrierDelay` Int32,
`WeatherDelay` Int32,
`NASDelay` Int32,
`SecurityDelay` Int32,
`LateAircraftDelay` Int32,
`FirstDepTime` String,
`TotalAddGTime` String,
`LongestAddGTime` String,
`DivAirportLandings` String,
`DivReachedDest` String,
`DivActualElapsedTime` String,
`DivArrDelay` String,
`DivDistance` String,
`Div1Airport` String,
`Div1AirportID` Int32,
`Div1AirportSeqID` Int32,
`Div1WheelsOn` String,
`Div1TotalGTime` String,
`Div1LongestGTime` String,
`Div1WheelsOff` String,
`Div1TailNum` String,
`Div2Airport` String,
`Div2AirportID` Int32,
`Div2AirportSeqID` Int32,
`Div2WheelsOn` String,
`Div2TotalGTime` String,
`Div2LongestGTime` String,
`Div2WheelsOff` String,
`Div2TailNum` String,
`Div3Airport` String,
`Div3AirportID` Int32,
`Div3AirportSeqID` Int32,
`Div3WheelsOn` String,
`Div3TotalGTime` String,
`Div3LongestGTime` String,
`Div3WheelsOff` String,
`Div3TailNum` String,
`Div4Airport` String,
`Div4AirportID` Int32,
`Div4AirportSeqID` Int32,
`Div4WheelsOn` String,
`Div4TotalGTime` String,
`Div4LongestGTime` String,
`Div4WheelsOff` String,
`Div4TailNum` String,
`Div5Airport` String,
`Div5AirportID` Int32,
`Div5AirportSeqID` Int32,
`Div5WheelsOn` String,
`Div5TotalGTime` String,
`Div5LongestGTime` String,
`Div5WheelsOff` String,

```
  `Div5TailNum` String
) ENGINE = MergeTree(FlightDate, (Year, FlightDate), 8192)
```

表结构与ontime完全一样。

建分布表

```
CREATE TABLE ontime_all AS ontime_local
ENGINE = Distributed(cluster_3shards_1replicas, default, ontime_local, rand())
```

分布表（Distributed）本身不存储数据，相当于路由，需要指定集群名、数据库名、数据表名、分片KEY，这里分片用rand()函数，表示随机分片。查询分布表，会根据集群配置信息，路由到具体的数据表，再把结果进行合并。

ontime_all与ontime在同一个节点上，方便插入数据。

插入数据

```
INSERT INTO ontime_all SELECT * FROM ontime;
```

把ontime的数据插入到ontime_all，ontime_all会随机插入到三个节点的ontime_local里。

表ontime需要提前建好，并导入数据。

导入完成后，查看总数据量：

```
hadoop1 :) select count(1) from ontime_all;

SELECT count(1)
FROM ontime_all

┌────count(1)─┐
│ 177920306 │
└─────────────────────┘
```

再看下每个节点的数据：

```
hadoop1 :) select count(1) from ontime_local;

SELECT count(1)
FROM ontime_local

┌──count(1)─┐
│ 59314494 │
└─────────────────┘
```

可以看到，每个节点大概有1/3的数据。

性能对比

对比一下分片与不分片的性能差异。

不分片：

```
hadoop1 :) select Carrier, count() as c, round(quantileTDigest(0.99)(DepDelay), 2) as q from ontime group by Carrier
order by q desc limit 5;

SELECT
    Carrier,
    count() AS c,
    round(quantileTDigest(0.99)(DepDelay), 2) AS q
FROM ontime
GROUP BY Carrier
ORDER BY q DESC
LIMIT 5

┌─Carrier─┬──────c─┬─────q─┐
│ B6      │ 2991782 │ 191.22 │
│ NK      │  412396 │ 190.97 │
│ EV      │ 6222018 │ 187.17 │
│ XE      │ 2145095 │ 179.55 │
```

```
| VX      |  371390 |  178.3 |
└─────────┴─────────┴────────┘

5 rows in set. Elapsed: 1.951 sec. Processed 177.92 million rows, 1.07 GB (91.18 million rows/s., 547.11 MB/s.)
```

用时1.951秒。

分片：

```
hadoop1 :) select Carrier, count() as c, round(quantileTDigest(0.99)(DepDelay), 2) as q from ontime_all group by Carrier order by q desc limit 5;

SELECT
    Carrier,
    count() AS c,
    round(quantileTDigest(0.99)(DepDelay), 2) AS q
FROM ontime_all
GROUP BY Carrier
ORDER BY q DESC
LIMIT 5

┌─Carrier─┬───────c─┬─────q─┐
| B6      | 2991782 |  191.2 |
| NK      |  412396 | 191.06 |
| EV      | 6222018 | 187.22 |
| XE      | 2145095 | 179.42 |
| VX      |  371390 | 178.33 |
└─────────┴─────────┴────────┘

5 rows in set. Elapsed: 0.960 sec. Processed 177.92 million rows, 1.07 GB (185.37 million rows/s., 1.11 GB/s.)
```

用时0.96秒，速度大约提升2倍。

现在，停掉一个节点，会是神马情况？

```
Received exception from server (version 18.10.3):
Code: 279. DB::Exception: Received from localhost:9000, 127.0.0.1. DB::NetException. DB::NetException: All connection tries failed. Log:

Code: 32, e.displayText() = DB::Exception: Attempt to read after eof, e.what() = DB::Exception
Code: 210, e.displayText() = DB::NetException: Connection refused: (hadoop2:9000, 192.168.0.16), e.what() = DB::NetException
Code: 210, e.displayText() = DB::NetException: Connection refused: (hadoop2:9000, 192.168.0.16), e.what() = DB::NetException
```

报错了，看来clickhouse的处理很严格，如果一个分片不可用，就整个分布式表都不可用了。

当然，此时如果查本地表ontime_local还是可以的。

那么，如何解决整个问题呢？这就是前文所说的稳定性问题了，解决方案是：数据备份！

数据备份

说明一点，数据备份与分片没有必然联系，这是两个方面的问题。但在clickhouse中，replica是挂在shard上的，因此要用多副本，必须先定义shard。

最简单的情况：1个分片多个副本。

添加集群

像之前一样，再配置一个集群，叫做cluster_1shards_2replicas，表示1分片2副本，配置信息如下：

```
<yandex>
        <!-- 1分片2备份 -->
        <cluster_1shards_2replicas>
            <shard>
```

```
            <internal_replication>false</internal_replication>
            <replica>
                <host>hadoop1</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>hadoop2</host>
                <port>9000</port>
            </replica>
        </shard>
    </cluster_1shards_2replicas>
</yandex>
```

注意，如果配置文件没有问题，是不用重启clickhouse-server的，会自动加载！

建数据表

```
CREATE TABLE `ontime_local_2` (
  ...
) ENGINE = MergeTree(FlightDate, (Year, FlightDate), 8192)
```

表名为ontime_local_2，在三台机器分别执行。

建分布表

```
CREATE TABLE ontime_all_2 AS ontime_local_2
ENGINE = Distributed(cluster_1shards_2replicas, default, ontime_local_2, rand())
```

表名是ontime_all_2，使用cluster_1shards_2replicas集群，数据为ontime_local_2。

导入数据

```
INSERT INTO ontime_all_2 SELECT * FROM ontime
```

查询

```
hadoop1 :) select count(1) from ontime_all_2;

SELECT count(1)
FROM ontime_all_2

┌────count(1)─┐
│ 177920306 │
└─────────────┘
```

查询ontime_local_2，两个节点都有全量数据。

关掉一个服务器，仍能查询全量数据，数据副本已经生效。

一致性

既然有多副本，就有个一致性的问题：加入写入数据时，挂掉一台机器，会怎样？

我们来模拟一下：

1. 停掉hadoop2服务

```
service clickhouse-server stop
```

1. 通过ontime_all_2插入几条数据

```
hadoop1 :) insert into ontime_all_2 select * from ontime limit 10;
```

1. 启动hadoop2服务

```
service clickhouse-server start
```

1. 查询验证

查看两个机器的ontime_local_2、以及ontime_all_2，发现都是总数据量都增加了10条，说明这种情况下，集群节点之间能够自动同步

再模拟一个复杂点的：

1. 停掉hadoop2；
2. 通过ontime_all_2插入10条数据；
3. 查询ontime_all_2，此时数据增加了10条；
4. 停掉hadoop1；

5. 启动hadoop2，此时，整个集群不可用；
6. 查询ontime_all_2，此时，集群恢复可用，但数据少了10条；
7. 启动hadoop1，查询ontime_all_2、ontime_local_2，数据自动同步；

上边都是通过ontime_all_2表插入数据的，如果通过ontime_local_2表插入数据，还能同步吗？

1. hadoop1上往ontime_local_2插入10条数据；
2. 查询hadoop2，ontime_local_2数据没有同步

综上，通过分布表写入数据，会自动同步数据；而通过数据表写入数据，不会同步；正常情况没什么大问题。

更复杂的情况没有模拟出来，但是可能会存在数据不一致的问题，官方文档描述如下：

Each shard can have the 'internal_replication' parameter defined in the config file.

If this parameter is set to 'true', the write operation selects the first healthy replica and writes data to it. Use this alternative if the Distributed table "looks at" replicated tables. In other words, if the table where data will be written is going to replicate them itself.

If it is set to 'false' (the default), data is written to all replicas. In essence, this means that the Distributed table replicates data itself. This is worse than using replicated tables, because the consistency of replicas is not checked, and over time they will contain slightly different data.

翻译下：

分片可以设置internal_replication属性，这个属性是true或者false，默认是false。

如果设置为true，则往本地表写入数据时，总是写入到完整健康的副本里，然后由表自身完成复制，这就要求本地表是能自我复制的。

如果设置为false，则写入数据时，是写入到所有副本中。这时，是无法保证一致性的。

举个栗子，一条数据要insert到ontime_all_2中，假设经过rand()实际是要写入到hadoop1的ontime_local表中，此时ontime_local配置了两个副本。

如果internal_replication是false，那么就会分别往两个副本中插入这条数据。注意！！！分别插入，可能一个成功，一个失败，插入结果不检验！这就导致了不一致性；

而如果internal_replication是true，则只往1个副本里写数据，其他副本则是由ontime_local自己进行同步，这样就解决了写入一致性问题。

虽然没有模拟出数据不一致的情况，实际中可能会遇到，所以官方建议使用表自动同步的方式，也就是internal_replication为true。

具体怎么用，下边具体介绍。

自动数据备份

自动数据备份，是表的行为，ReplicatedXXX的表支持自动同步。

Replicated前缀只用于MergeTree系列（MergeTree是最常用的引擎），即clickhouse支持以下几种自动备份的引擎：

ReplicatedMergeTree

ReplicatedSummingMergeTree

ReplicatedReplacingMergeTree

ReplicatedAggregatingMergeTree

ReplicatedCollapsingMergeTree

ReplicatedGraphiteMergeTree

再强调一遍，Replicated表自动同步与之前的集群自动同步不同，是表的行为，与clickhouse_remote_servers配置没有关系，只要有zookeeper配置就行了。

为了说明这个问题，先不配置clickhouse_remote_servers，只添加zookeeper配置：

```
<yandex>
    ...
    <zookeeper-servers>
        <node index="1">
            <host>hadoop1</host>
            <port>2181</port>
        </node>
        <node index="2">
            <host>hadoop2</host>
            <port>2181</port>
        </node>
```

```
        <node index="3">
            <host>hadoop3</host>
            <port>2181</port>
        </node>
    </zookeeper-servers>
    ...
</yandex>
```

建数据表

hadoop1:

```
CREATE TABLE `ontime_replica` (
  ...
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/ontime', 'replica1', FlightDate, (Year, FlightDate), 8192);
```

hadoop2:

```
CREATE TABLE `ontime_replica` (
  ...
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/ontime', 'replica2', FlightDate, (Year, FlightDate), 8192);
```

hadoop3:

```
CREATE TABLE `ontime_replica` (
  ...
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/ontime', 'replica3', FlightDate, (Year, FlightDate), 8192);
```

查看zk信息：

```
[zk: localhost:2181(CONNECTED) 0] ls /clickhouse/tables/ontime/replicas
[replica2, replica3, replica1]
```

可以看到，zk中已经有了对应的路径和副本信息。

插入数据

hadoop1:

```
INSERT INTO ontime_replica SELECT * FROM ontime
```

注意！只在一个机器上执行插入操作。

查看数据

分别在三个机器上查询ontime_replica，都有数据，且数据完全一样。

可以看到，这种方式与之前方式的区别，直接写入一个节点，其他节点自动同步，完全是表的行为；而之前的方式必须创建Distributed表，并通过Distributed表写入数据才能同步（目前我们还没有给ontime_replica创建对应的Distributed表）。

配置集群

```
...
        <!-- 1分片3备份：使用表备份-->
        <cluster_1shards_3replicas>
            <shard>
                <internal_replication>true</internal_replication>
                <replica>
                    <host>hadoop1</host>
                    <port>9000</port>
                </replica>
                <replica>
                    <host>hadoop2</host>
                    <port>9000</port>
                </replica>
                <replica>
                    <host>hadoop3</host>
                    <port>9000</port>
```

```
            </replica>
        </shard>
    </cluster_1shards_3replicas>
...
```

集群名字为cluster_1shards_3replicas，1个分片，3个副本；与之前不同，这次设置internal_replication为true，表示要用表自我复制功能，而不用集群的复制功能。

建分布表

```
CREATE TABLE ontime_replica_all AS ontime_replica
ENGINE = Distributed(cluster_1shards_3replicas, default, ontime_replica, rand())
```

表名为ontime_replica_all，使用cluster_1shards_3replicas集群，数据表为ontime_replica。

查询分布表：

```
hadoop1 :) select count(1) from ontime_replica_all;

SELECT count(1)
FROM ontime_replica_all

┌───count(1)──┐
│ 177920306 │
└─────────────┘
```

分布表写入

前边说了，一个节点ontime_replica写入数据时，其他节点自动同步；那如果通过分布表ontime_replica_all写入数据会如何呢？

其实，前文已经提到过，internal_replication为true，则通过分布表写入数据时，会自动找到“最健康”的副本写入，然后其他副本通过表自身的复制功能同步数据，最终达到数据一致。

分片 + 备份

分片，是为了突破单机上限（存储、计算等上限），备份是为了高可用。

只分片，提升了性能，但是一个分片挂掉，整个服务不可用；只备份，确实高可用了，但是整体还是受限于单机瓶颈（备份1000份与备份2份没什么区别，除了更浪费机器）。

所以，生产中这两方面需要同时满足。

其实，只要把之前的分片和备份整合起来就行了，例如，3分片2备份的配置如下：

```
...
        <!-- 3分片2备份：使用表备份 -->
        <cluster_3shards_2replicas>
            <shard>
                <internal_replication>true</internal_replication>
                <replica>
                    <host>hadoop1</host>
                    <port>9000</port>
                </replica>
                <replica>
                    <host>hadoop2</host>
                    <port>9000</port>
                </replica>
            </shard>
            <shard>
                <internal_replication>true</internal_replication>
                <replica>
                    <host>hadoop3</host>
                    <port>9000</port>
                </replica>
                <replica>
                    <host>hadoop4</host>
```

```
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <internal_replication>true</internal_replication>
            <replica>
                <host>hadoop5</host>
                <port>9000</port>
            </replica>
            <replica>
                <host>hadoop6</host>
                <port>9000</port>
            </replica>
        </shard>
    </cluster_3shards_2replicas>
...
```

这里一共需要建6个数据表：

```
CREATE TABLE `ontime_replica` (
  ...
) ENGINE = ReplicatedMergeTree('/clickhouse/tables/ontime/{shard}', '{replica}', FlightDate, (Year, FlightDate), 8192);
```

其中，{shard}和{replica}是macros配置（相当于环境变量），修改配置文件：

```
vi /etc/clickhose-server/metrika.xml
```

添加内容：

```
...
    <macros>
        <shard>01</shard>
        <replica>01</replica>
    </macros>
...
```

每台机器的shard和replica值根据具体情况设置，例如，这里是3分片2副本，则配置如下：

```
hadoop1: shard=01, replica=01
hadoop2: shard=01, replica=02
hadoop3: shard=02, replica=01
hadoop4: shard=02, replica=02
hadoop5: shard=03, replica=01
hadoop6: shard=03, replica=02
```

使用macros只是为了建表方便（每个机器可以使用同样的建表语句），不是必须的，只要ReplicatedMergeTree指定zk路径和replica值即可。

由于资源有限，这里不实验了。

需要提醒一下，每个clickhouse-server实例只能放一个分片的一个备份，也就是3分片2备份需要6台机器（6个不同的clickhouse-server）。

之前为了节省资源，打算循环使用，把shard1的两个副本放到hadoop1、hadoop2两个机器上，shard2的两个副本放到hadoop2、hadoop3上，shard3的两个副本放到hadoop3、hadoop1上，结果是不行的。

原因是shard+replica对应一个数据表，Distributed查询规则是每个shard里找一个replica，把结果合并。

假如按照以上设置，可能一个查询解析结果为：

取shard1的replica1，对应hadoop1的ontime_replica；

取shard2的replica2，对应hadoop3的ontime_replica；

取shard3的replica2，对应hadoop1的ontime_replica；

最后，得到的结果是hadoop1的ontime_replica查询两次+hadoop3的ontime_replica查询一次，结果是不正确的。

由于之前用elasticsearch集群，可以用3台服务器来创建5分片2备份的索引，所以想当然的认为clickhouse也可以，结果坑了两天。