# 最强OLAP分析引擎-Clickhouse快速精通二

==楼兰==



# 四、集群机制

这一章来分享clickhouse的集群机制。其实clickhouse的单机性能通常已经非常优秀了，底层的数据压缩效率是很高的。例如之前看到过，官方提供的Github Evnets数据集三十多亿数据，用200G硬盘也就存下来了，这种配置要求，在服务器级别是压力不大的。另外，单机查询性能如果CPU配置足够好的话，查询速度也基

本可以达到秒级。

clickhouse提供了集群机制的支持。clickhouse的集群主要有两个作用，一是数据副本，也就是将数据冗余到另外的机器上，用以保证高可用。二是分布式表，也就是将一个表的数据分散到多个节点上保存。

在这其中，数据副本是相对比较重要的，而至于分布式表，大部分场景下其实是不需要的。这是因为clickhouse的单机性能通常已经非常优秀了，底层的数据压缩效率是很高的。例如之前看到过，官方提供的Github Evnets数据集三十多亿数据，用200G硬盘也就存下来了，这种配置要求，在服务器级别是压力不大的。另外，单机查询性能如果CPU配置足够好的话，查询速度也基本可以达到秒级。分布式表反而会消耗网络资源，降低查询速度。

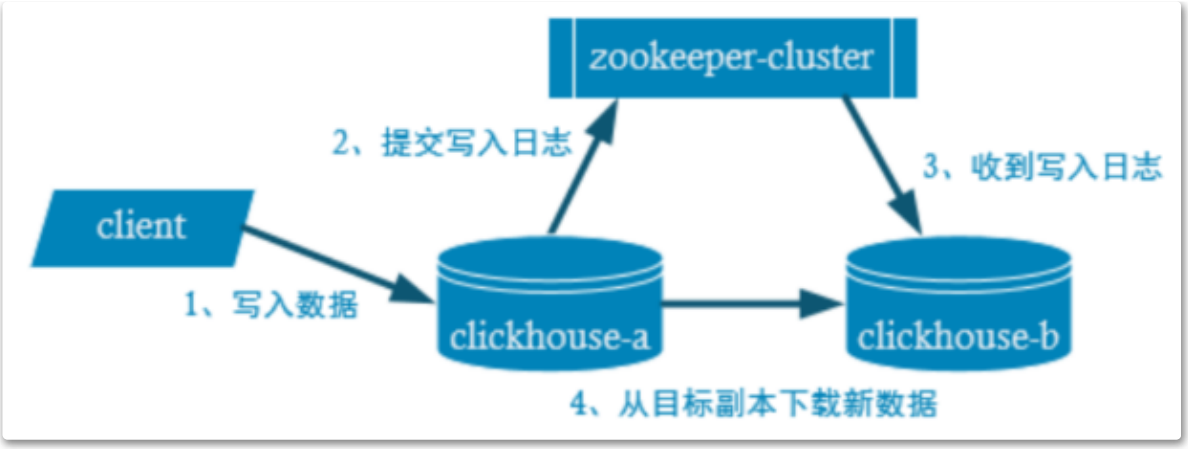
# 4.1 数据副本

副本的目的是保障数据的高可用。当一台clickhouse节点宕机了，也可以从其他备份服务器获得相同的数据。clickhosue只有MergeTree系列的表引擎可以支持副本。针对MergeTree系列的表引擎，clickhouse都提供了对应的Replicated*MergeTree表引擎来支持数据副本。

> 具体参看官方文档：https://clickhouse.com/docs/zh/engines/table-engines/mergetree-family/replication/

clickhouse的数据副本机制是表级别的，也就是说只针对表进行复制，一个数据库中可以同时包含复制表和非复制表。副本机制对于select查询是没有影响的，查询复制表和非复制表的速度是一样的。而写入数据时，clickhouse的集群没有主从之分，大家都是平等的。只不过配置了复制表后，insert以及alter操作会同步到对应的副本机器中。对于复制表，每个Insert语句会往Zookeeper中写入十来条记录，相比非复制表，写ZK会导致Insert语句的延迟时间略长。但是，在clickhouse建议的每秒不超过一个Insert语句的执行频率下，这个延迟时间不会有太大的影响。

集群数据写入流程如下：




在这其中，其实还有很多的实现细节。数据副本需要经过网络传输，所以副本中写入数据是有延迟的。默认情况下，clickhouse对于Insert语句，只会等待一个副本写入成功后就会返回。如果有多个副本的情况下，clickhouse是有可能丢失数据的。写入数据时，clickhouse只保证单个数据块的写入是原子的，而不能保证所有的数据写入是原子的。一个数据块的大小可以根据max_insert_block_size=1048576行进行分块。数据块写入时是会去重的，一个同样的Insert语句多次重复执行，数据库块只会执行一次。这是为了防止用户重复插入数据或者网络波动等原因造成的数据重发。这个去重机制只对应Replicated*MergeTree系列的表引擎，普通的MergeTree是不带这个去重功能的。

**clickhouse数据副本使用实例：**

1、启动一个Zookeeper集群。

clickhouse的集群搭建依赖zookeeper。如果没有配置zookeeper的话，依然可以创建复制表，但是这些复制表都将变成只读。另外，官方建议，不要在clickhouse所在的服务器上运行zookeeper。因为zookeeper对数据延迟非常敏感，而clickhouse可能会占用所有可用的系统资源。

注意当前版本的clickhouse要求zookeeper版本不低于3.4.5。并且，官方对zookeeper的优化配置也提出了指导意见。具体参见官方文档：https://clickhouse.com/docs/zh/operations/tips/

2、在clickhouse中进行配置。

打开clickhouse的配置文件 /etc/clickhouse-server/config.xml，在配置文件中指定你的zookeeper集群地址，然后重启服务器。在730行左右找到标签进行配置：

```
<zookeeper>
    <node>
```

```
        <host>hadoop01</host>
        <port>2181</port>
    </node>
    <node>
        <host>hadoop02</host>
        <port>2181</port>
    </node>
    <node>
        <host>hadoop03</host>
        <port>2181</port>
    </node>
</zookeeper>
```

3、创建复制表

建表的方式与之前使用MergeTree系列表引擎基本都是一样的，只不过在Engine上要加上Replicated。

```
CREATE TABLE t_stock_replicated
(
    `id` UInt32,
    `sku_id` String,
    `total_amount` Decimal(16, 2),
    `create_time` DateTime
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/t_stock','hadoop01')
PARTITION BY toYYYYMMDD(create_time)
PRIMARY KEY id
ORDER BY (id, sku_id);
```

在ReplicatedMergeTree中传入两个参数，一个参数是zk的路径。一个参数是备份的名字。

**在多个节点中，同一个集群的复制表，他们的zk路径应该是一样的，但是备份名应该是不相同的。**

> 在这两个参数中，支持使用占位符例如 /clickhouse/tables/{layer}-{shard}/t_stock 和 ${replicate}。这些占位符的取值都在config.xml中配置的 宏配置这一片段当中。打开注释即可。

```
<macros>
 <layer>05</layer>
 <shard>02</shard>
 <replica>example05-02-1.yandex.ru</replica>
</macros>
```

例如之前的clickhouse数据库是安装在hadoop01机器上。完成上述的操作之后。在hadoop02机器上，同样安装一个clickhouse服务，config.xml文件同步配置。hadoop02机器上只需要创建一个表。

```
CREATE TABLE t_stock_replicated
(
    `id` UInt32,
    `sku_id` String,
    `total_amount` Decimal(16, 2),
    `create_time` DateTime
)
ENGINE = ReplicatedMergeTree('/clickhouse/tables/t_stock','hadoop02')
PARTITION BY toYYYYMMDD(create_time)
PRIMARY KEY id
ORDER BY (id, sku_id);
```

那么两个机器上的t_stock_replicated表的数据就会同步。而且这个数据同步是双向的，也就是说写入到hadoop02机器上的数据，同样会同步到hadoop01机器上。

同样的方式，你可以将clickhouse中的数据复制到更多的集群中。在这个过程中，你基本不需要担心zookeeper的性能问题。一个Zookeeper集群能给整个clickhouse集群支撑协调每秒几百个INSERT，数据的吞吐量可以跟不用复制的数据一样高。官方给出的Yandex.Metrica集群，大约有300台服务器，依然一个zookeeper搞定了。

clickhouse这种集群机制相比很多数据库产品简直不要太简单了。

## 4.2 分布式表

副本机制能够提高数据的可用性，降低丢失数据的风险，但是每台服务器上都需要容纳全量的数据，没有解决数据的横向扩容的问题。在clickhouse中，可以通过水平切分的方式，将完整的数据集切分成不同的分片。这些分片只保存一部分数据，分布在不同的节点上。然后再通过Distributed表引擎将数据拼接起来作为一个完整的表使用。Distributed表引擎本身不存储数据，就有点像ShardingSphere与

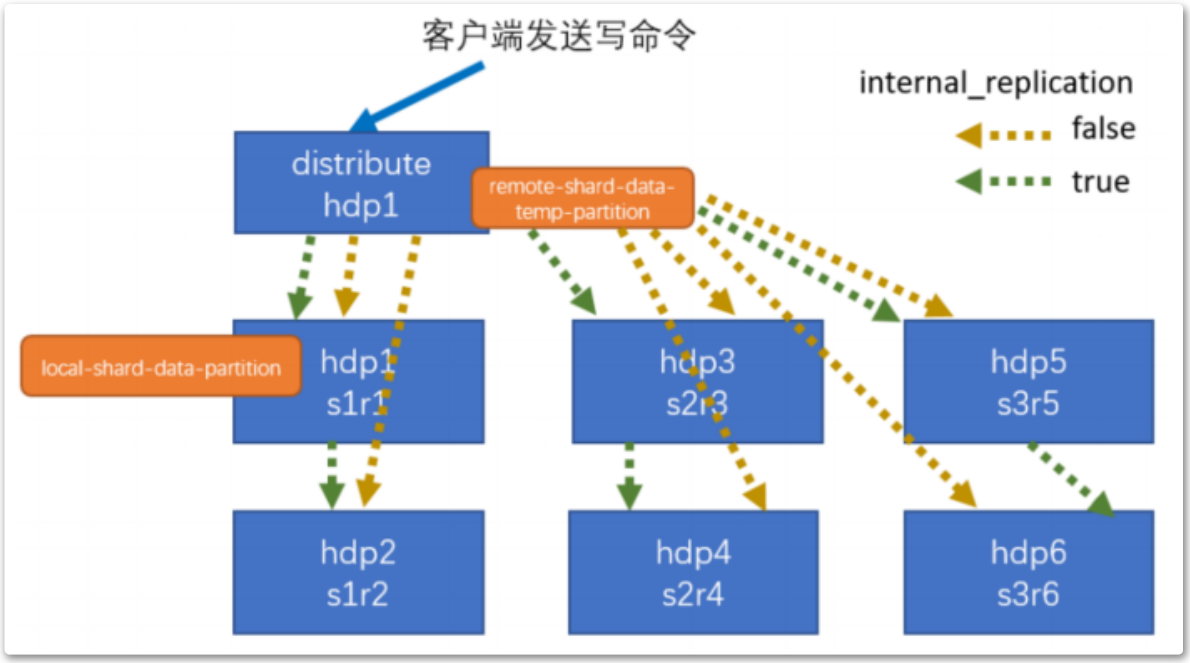
MySQL，只是一个中间件，通过分布式逻辑表来写入、分发、路由操作多台节点不同分片的分布式数据。

之前分析过，clickhouse的单机性能是很强的。很多企业在实际运用过程中，会配置副本机制来做高可用，但是通常不会做分片，这样可以降低查询时的性能消耗。

> 具体参看官方文档： https://clickhouse.com/docs/zh/engines/table-engines/special/distributed/

使用分片表需要先配置集群。一个集群由多个分片shard，每个分片下可以配置一个或多个副本replica。这些replica副本可以配置成数据副本，当然这也不是必须的。分片表是可以和数据副本混合使用的。在声明分片表时需要指定一个集群。

数据分片之后，写入的流程是这样的

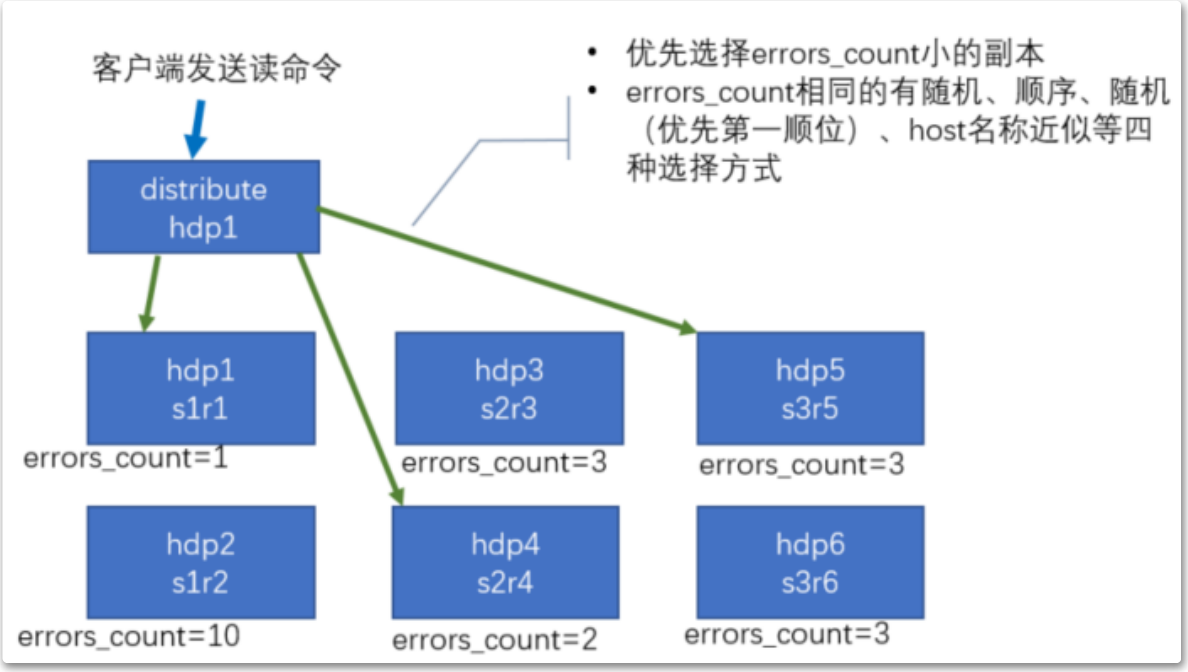



配置分片时可以指定一个 internal_replication 参数。为true时，写操作只选一个正常的副本写入数据。如果分片内的子表都是复制表(Replicated*MergeTree引擎)，那就请使用这个配置。如果参数设置为false,写操作就会将数据写入所有副本。实际上，这种方式还不如使用复制表更好。因为这种机制不会检查副本的一致性，并且随着时间推移，副本数据可能会有些不一致。

在配置集群时，会给每个分片shard分配一个权重weight，默认是1.数据会一句分片的权重按比例分发到对应的分片上。这样，你可以选择在配置比较好的机器上分配更多的权重。

另外要注意，分布式表的数据也是异步写入的。对于分布式表的INSERT，数据块只写本地文件系统。之后会尽快地在后台发送给远程服务器。如果在INSERT的时候机器节点丢失了，则会造成插入的数据丢失，并且不会再进行后续的检查。而之前介绍的数据副本机制，会在机器服务恢复过来后，继续同步之前丢失的数据。

数据读取的流程是这样的

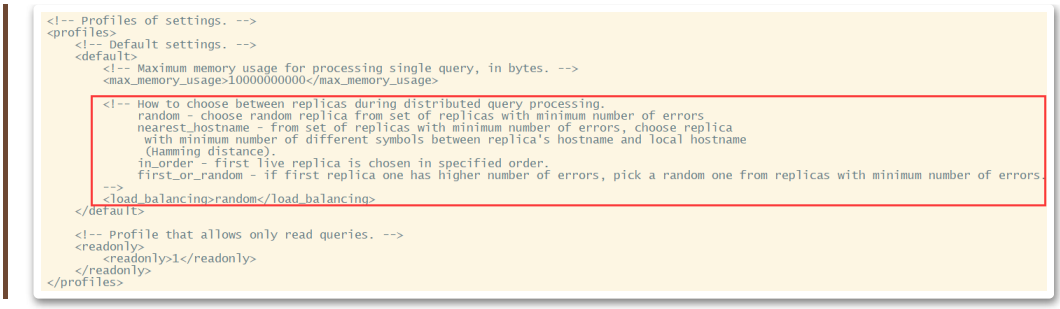



通过分布式引擎，可以像使用本地服务器一样使用集群。但是，集群不是自动扩展的，需要修改配置文件，再重启服务。使用分布式引擎时，如果集群信息发生了变动，分布式表中的集群信息也会即时更新，无需重启服务器。但是，通常分布式表的集群结构是不建议经常变动的，如果集群结构不稳定的话，这时就不建议使用分布式表了，可以使用远程表函数remote来访问。

SELECT查询会被发送到所有分片，并且无论数据在分片中如何分布(即使数据完全随机分布)都可以正常工作。添加新分片时，也不必将旧数据传输到该分片。你可以给新分片分配较大的权重然后写新数据，数据可能会一段时间分布不均，但是查询会正常高效的运行。

对于同一个分片下的多个副本，分布式表肯定只能读取其中的一个副本。那如何选择要读取数据的副本呢？在服务端会维护一个errors_count，也就是读取当前副本出现错误的次数。默认情况下，clickhouse会优先选择errors_count出错次数比较小的副本来读取。

> 关于副本选择机制，参见clickhouse的配置文件users.xml




### clickhouse分布式表使用实例

1、同样需要zookeeper支持。

2、在clickhouse中配置集群。

同样是打开clickhouse的配置文件 /etc/clickhouse-server/config.xml，在配置文件中配置一个分布式集群，然后重启服务器。在590行左右找到标签进行配置：

```
<remote_servers>
    <logs>
        <shard>
            <!-- Optional. Shard weight when writing data. Default: 1. -->
            <weight>1</weight>
            <!-- Optional. Whether to write data to just one of the
replicas. Default: false (write data to all replicas). -->
            <internal_replication>false</internal_replication>
            <replica>
                <host>hadoop01</host>
                <port>9000</port>
            </replica>
        </shard>
        <shard>
            <weight>2</weight>
            <internal_replication>false</internal_replication>
            <replica>
                <host>hadoop02</host>
                <port>9000</port>
            </replica>
        </shard>
    </logs>
</remote_servers>
```

> 配置文件中默认已经配置了几个分布式集群，生产上建议删掉这段配置。实验中最好注释掉。

在这一段配置中，配置了一个名为logs的集群，他由两个分片shard组成，每个分片只包含一个副本replica。在标签中配置包含副本的所有机器节点(实际使用时当然还是建议配置多个副本)。两个分片所包含的副本节点不要求相同，实际使用时也只有不相同才有意义。在标签中，除了已经配置过的这些外，还可以配置一些子标签。

- port： 消息传递的TCP端口。由config.xml中的<tcp_port>属性配置，默认是9000。另外还一个http_port配置的是8123，这两个不要搞混了
- user 这个标签是用于连接远程clickhouse服务器的用户名。默认是default。在users.xml中可以进行更多的用户权限配置。如果配置了其他用户，这个标签就必须添加。
- password 这个标签是用于连接远程clickhouse服务器的密码。默认是空字符串。我们的实验环境中，当初安装clickhouse服务时并没有设定密码，所以不用添加。
- secure 配置是否使用ssl进行连接。默认true
- compression: 是否使用数据压缩，默认true。

配置集群时，集群名字不能带点号。配置文件需要在集群服务上进行同步。

配置的集群可以在System.cluster表中查看到。也可以通过show clusters中看到。

> 在我们的实验中为了专注于分布式表引擎，在每个shard下只配置了一个副本replica。实际上，clickhouse的每个分片下都支持配置多个副本replica。具体参见配置文件中的示例。

3、使用分片表

在建表时，通过 on cluster 关键字指定集群名

```
CREATE TABLE t_stock_local on cluster logs
(
    `id` UInt32,
    `sku_id` String,
    `total_amount` Decimal(16, 2),
    `create_time` DateTime
)
-- ENGINE =
ReplicatedMergeTree('/clickhouse/tables/{shard}/t_stock','{replica}')
ENGINE = MergeTree()
PARTITION BY toYYYYMMDD(create_time)
PRIMARY KEY id
ORDER BY (id, sku_id);
```

在hadoop01机器上执行这个建表语句，这个表就会同步到hadoop02机器上。

```
CREATE TABLE t_stock_local ON CLUSTER logs
(
    `id` UInt32,
    `sku_id` String,
    `total_amount` Decimal(16, 2),
    `create_time` DateTime
)
ENGINE = MergeTree
PARTITION BY toYYYYMMDD(create_time)
PRIMARY KEY id
ORDER BY (id, sku_id)

Query id: a4ab8df0-0772-40b8-b1ec-9f0879ca1a74

┌─host─────┬─port─┬─status─┬─error─┬─num_hosts_remaining─┬─num_hosts_active─┐
│ hadoop02 │ 9000 │      0 │       │                   1 │                0 │
│ hadoop01 │ 9000 │      0 │       │                   0 │                0 │
└──────────┴──────┴────────┴───────┴─────────────────────┴──────────────────┘
```

这里要注意下，我们示例中每个分片只有1个分片，并没有配置副本。通常还是建议在一个分片下建立多个副本，在多个副本上建立复制表。但是这时，之前配置副本时说到不同节点上的replica副本名称不可以相同。这时，如果replica副本名称还是在SQL语句中指定的话，就会造成两边的副本名重复了。这时的做法就是将副本名改为使用配置文件中配置的macros宏，在两台机器上将副本名称区分开。

## 4、使用Distributed表引擎创建分布式表

```
CREATE TABLE t_stock_distributed on cluster logs
(
    `id` UInt32,
    `sku_id` String,
    `total_amount` Decimal(16, 2),
    `create_time` DateTime
)
ENGINE = Distributed(logs,default, t_stock_local,hiveHash(sku_id));
```

在Destributed引擎中，需要指定几个参数，依次是： 集群名称、库名、本地表名、分片键。其中分片键必须是整型数字，可以使用rand()来返回一个随机数字，也可以是id这样一个int字段，如果id的数据分布不够均匀，也可以使用intHash64()函数进行一下散列。而对于sku_id，他是string类型的，所以要用 hiveHash函数转换一下。如果是随机分片可以用rand()。

后续就可以像使用一个普通表一样使用这个分布式表了。数据会分开存储到不同分片的t_stock_local表当中。

例如从下图中可以看到，在hadoop01机器上的t_stock_distributed表中可以查到数据，但是在t_stock_local表中就没有数据。而数据全部写到了hadoop02机器上的t_stock_local中。hadoop02机器上的分片有更大的权重。

```
SELECT *
FROM t_stock_distributed

Query id: 0784a943-e9e4-4799-b61a-7abd6a55a265

┌──id─┬─sku_id─┬─total_amount─┬─────────create_time─┐
│ 105 │ sku_002 │          600 │ 2020-06-04 12:00:00 │
└─────┴─────────┴──────────────┴─────────────────────┘

┌──id─┬─sku_id─┬─total_amount─┬─────────create_time─┐
│ 104 │ sku_002 │        12000 │ 2020-06-03 13:00:00 │
└─────┴─────────┴──────────────┴─────────────────────┘

┌──id─┬─sku_id─┬─total_amount─┬─────────create_time─┐
│ 101 │ sku_002 │         2000 │ 2020-06-01 11:00:00 │
│ 101 │ sku_002 │         2000 │ 2020-06-01 14:00:00 │
│ 102 │ sku_004 │         2500 │ 2020-06-01 12:00:00 │
│ 102 │ sku_004 │         2500 │ 2020-06-01 15:00:00 │
│ 103 │ sku_002 │         2000 │ 2020-06-01 16:00:00 │
│ 104 │ sku_002 │        12000 │ 2020-06-01 17:00:00 │
└─────┴─────────┴──────────────┴─────────────────────┘

┌──id─┬─sku_id─┬─total_amount─┬─────────create_time─┐
│ 103 │ sku_002 │         2000 │ 2020-06-02 13:00:00 │
└─────┴─────────┴──────────────┴─────────────────────┘

9 rows in set. Elapsed: 0.032 sec.

hadoop01 :) select * from t_stock_local;

SELECT *
FROM t_stock_local

Query id: df35634a-76b8-43b6-8ce8-5da3abd6075b

Ok.

0 rows in set. Elapsed: 0.020 sec.
```

最后注意，使用clickhouse，通常只建议使用数据副本，不建议使用分布式表！

# 五、配置优化

这一章主要分享如何优化clickhouse的配置。

对于clickhouse的配置优化，最为重要的，也就是对服务资源分配的优化。而对于clickhouse来说，消耗最多的系统资源其实就是CPU，CPU使用率也是对clickhouse服务进行监控的一个很重要的指标。**CPU使用率一般达到50%左右会出现查询波动，达到70%会出现大范围的查询超时。**所以通常建议clickhouse服务要进行单独的部署，尽量不要跟其他服务共用服务器。相比而言，对于内存和磁盘，由于clickhouse的数据压缩效率是非常高的，所以通常不会形成性能瓶颈。clickhouse极简化的设计方式，使得这些配置项相比其他数据库是得到非常多的精简的。并且大部分的配置信息，clickhouse也都给出了默认值，大部分场景下，这些默认值都是最优化的。例如，对于内存，clickhouse并不像flink这类框架一样，设定框架内存、任务内存等等各种各样的内存参数。只在users.xml中设定了单个任务使用的内存上限，设置的值也是简单粗暴的10000000000 ， 10G。具体配置在users.xml中 -> -> <max_memory_usage> 参数。

clickhouse的配置文件集中在config.xml和users.xml两个配置文件里。config.xml的配置项参见官方文档：https://clickhouse.com/docs/zh/operations/server-configuration-parameters/settings/ users.xml的配置项参见官方文档：https://clickhouse.com/docs/zh/operations/settings/settings-users/ 。

其中config.xml主要是服务端参数，主要包含一些在session和query级别无法修改的参数，例如服务的集群配置，zookeeper配置等。而users.xml则是运行时参数，大部分可能对session和query产生影响。当然，这种拆分方式在clickhouse中其实并不是很严格。

对于运行时参数，有几个地方可以配置，读取的优先级依次如下：

- users.xml配置文件中。主要配置在 标签中，以不同配置文件的方式进行配置。默认提供了一个default的profile。另外，clickhouse也可以自行指定外部的配置文件。
- Session配置。在一个客户端中执行 SET [settings] = [value] 语句的方式指定。只对当前会话生效。
- 查询时指定。查询时可以有多个方式指定参数。HTTP接口可以直接在后面添加参数。使用clickhouse-client脚本的--settings=value 参数方式指定。或者在select 语句中直接指定。

下面列出clickhouse使用过程中比较有用的几个配置：

- background_pool_size 后台线程池的大小。默认16，实际生产中可以修改为CPU个数的2倍。
- max_concurrent_queries：最大并发请求数，默认值 100 。 通常不建议调整，如果并发要求高，可以以50为阶段调整。
- max_memory_usage： 单次查询使用的最大内存大小。默认100G。如果服务器的内存资源很丰富，可以适量调大一点。
- default_session_timeout ：默认session断开时间，默认60秒。另外还有max_session_timeout。最大session断开时间，默认3600秒。需要控制客户端连接时长时可以定制。
- http_port： http接口访问端口，默认8123。clickhouse最重要的端口。不光在HTTP接口中用，jdbc,obdc驱动也使用这个端口。与Grafana等监控平台集成也可以通过这个端口。
- mysql_port：默认值9004。clickhouse在这个端口上，将会假装成一个mysql数据库。直接使用mysql的jdbc驱动包就可以连接。
- postgresql_port：默认值9005。clickhouse在这个端口上，将会假装成一个postgresql数据库。
- tcp_port： TCP协议与客户端通信的端口自。默认9000
- interserver_http_port： 默认9009。 clickhouse内部通信的端口。

# 六、查询优化

## 6.1 查看执行计划

执行计划是进行查询调优的重要参考。clickhouse中，可以使用explain语句很方便的查看SQL语句的执行计划。完整的explain使用语法如下：

```
EXPLAIN [AST | SYNTAX | PLAN | PIPELINE] [setting = value, ...] SELECT ...
[FORMAT ...]
```

其中

- AST： 查看抽象语法树

  支持查看所有类型的语句，不光是SELECT语句。

  ```
  explain ast select number from system.numbers limit 10;
  ```

- SYNTAX：查询优化后的SQL语句

```
EXPLAIN SYNTAX SELECT * FROM system.numbers AS a, system.numbers AS b,
system.numbers AS c;
```

- PLAN： 用于查看执行计划。

  可以指定五个参数

  - header: 打印各个执行步骤的header说明。默认0
  - description: 打印执行步骤。默认1
  - indexes: 显示索引使用情况。默认0。只对MergeTree表引擎有用。
  - actions: 打印执行计划的详细信息。默认0.
  - json: 以JSON格式打印执行计划。默认0.

```
EXPLAIN SELECT sum(number) FROM numbers(10) GROUP BY number % 4;
EXPLAIN json = 1, description = 0 SELECT 1 UNION ALL SELECT 2 FORMAT
TSVRaw;
```

- PIPELINE： 用于查看pipeline计划。

  可以指定三个参数：

  - header： 打印各个步骤的header信息。默认0
  - graph: 打印以DOT图形语法描述的结果。默认0
  - compact: 如果开启了graph，紧凑打印行。默认开启。

```
EXPLAIN PIPELINE SELECT sum(number) FROM numbers_mt(100000) GROUP BY
number % 4;
```

# 6.2 clickhouse内置的语法优化规则

clickhouse底层提供了基于规则的SQL优化实现，会对一些低效的查询语句自动进行优化。这些优化的方式实际上也是我们写高效查询的一些指导。

1、COUNT优化

在调用count时，如果使用的是count()或者count(*)，且没有where条件时，会直接使用system.tables的total_row。

```
hadoop01 :) EXPLAIN SELECT count()FROM datasets.hits_v1;

EXPLAIN
SELECT count()
FROM datasets.hits_v1

Query id: cdc0be1b-d44f-4351-84d3-e816906ed128

┌─explain─────────────────────────────────────────────┐
│ Expression ((Projection + Before ORDER BY))         │
│   MergingAggregated                                 │
│     ReadFromPreparedSource (Optimized trivial count)│
└─────────────────────────────────────────────────────┘

3 rows in set. Elapsed: 0.013 sec.
```

2、聚合计算外推

```
explain syntax select sum(UserID*2) from datasets.visits_v1;
优化为
SELECT sum(UserID) * 2 FROM datasets.visits_v1
```

3、谓词下推

当group by有having子句，但是没有with cube、with rollup或者with totals修饰的时候，having条件会提前到where过滤。

谓词下推优化规则涉及到一个参数enable_optimize_predicate_expression。默认是1，打开谓词下推。设置为0就会关闭谓词下推优化。

```
EXPLAIN SYNTAX SELECT UserID FROM datasets.hits_v1 GROUP BY UserID HAVING
UserID = '8585742290196126178';
优化为
SELECT UserID FROM datasets.hits_v1 WHERE UserID = '8585742290196126178'
GROUP BY UserID
```

4、三元运算优化

三元运算符会被替换成为multif函数。需要设定一个参数optimize_if_chain_to_multiif=1

```
EXPLAIN SYNTAX SELECT number = 1 ? 'yes' : (number = 2 ? 'no' : 'inknown')
FROM numbers(10) settings optimize_if_chain_to_multiif = 1;
优化为
SELECT multiIf(number = 1, 'yes', number = 2, 'no', 'inknown') FROM
numbers(10) SETTINGS optimize_if_chain_to_multiif = 1
```

5、聚合函数消除

如果对聚合建，也就是group by 使用min\max\any等无意义的聚合函数，则会将这些聚合函数消除掉

```
EXPLAIN SYNTAX
SELECT
     sum(UserID * 2),
     max(VisitID),
     max(UserID)
FROM visits_v1
GROUP BY UserID
//返回优化后的语句
SELECT
     sum(UserID) * 2,
     max(VisitID),
     UserID
FROM visits_v1
GROUP BY UserID
```

# 6.3 高性能查询优化

## 1、选择合适的表引擎

虽然MergeTree是clickhouse中最为常用的表引擎，但是也不意味着MergeTree适合所有的场景。

## 2、建表时不要使用Nullable

Nullable类型虽然在处理空值问题时非常简单好用，但是，官方已经指出Nullable类型几乎总是会拖累性能，所以要尽量少用。在实际项目中，尽量使用字段的默认值表示空，或者自行指定一个在业务中无意义的值。

这是因为存储Nullable列时需要创建一个额外的文件来存储NULL的标记，并且Nullable列无法被索引。

## 3、合适的划分分区和索引

实际使用中，Partition by分区基本上是必须的。在划分分区时，通常建议按天分区。如果自行分区的话，单分区的数据最好不要超过一百万行。

## 4、数据变更优化

对clickhouse数据的增删改操作都会产生新的临时分区，会给MergeTree带来额外的合并任务。因此，数据变更操作不宜太频繁，这样会产生非常多的临时分区。一次操作的数据也不能太快。临时分区写入过快会导致Merge速度跟不上而报错。

官方一般建议一秒钟发起一次左右的写入操作，每次操作写入的数据量保持在2W~5W之间。具体根据服务器性能而定。

# 5、使用Prewhere替代where

clickhouse还提供了一个Prewhere关键字。他的用法与where基本上一样的。但是不同之处在于prewhere只支持*MergeTree系列引擎。他会先读取指令的列数据，来判断数据过滤。等待数据过滤完成之后再读取select声明的列字段来补全其他属性。与之对比，where语句是读取整行各个列的数据，再进行过滤，IO性能明显下降。

默认情况下，clickhouse就会使用prewhere语句代替where。

```
1  explain syntax select WatchID from datasets.hits_v1 where
   UserID='3198390223272470366';
2  优化为:
3  select WatchID from datasets.hits_v1 prewhere UserID='3198390223272470366';
4
```

> 有一参数 set optimize_move_to_prewhere=0; 可以关闭where自动转化为prewhere的优化规则。但是通常没有必要。

# 6、指定列和分区

首先 在数据量很大时，应该避免使用select *。 而应该指定具体需要查询的列名。并且列应该越少越好。

这很好理解，因为clickhouse是以列来存储数据的。查询的列越少，需要读取的文件就越少。消耗的IO资源减少了，性能自然就提高了。

然后 在查询分区表时，应该尽量在where条件中指定分区键的查询条件。

clickhouse中的分区实际上对应一个本地目录。指定分区键的查询条件，同样可以减少查询所需要遍历的数据文件，减少IO资源消耗。

# 7、避免构建虚拟列

如非必要，尽量直接使用clickhouse中的表已有的列，不要将计算结果构建成不存在的虚拟列。这样会非常消耗资源，浪费性能。通常情况下，都可以在数据进入clickhouse之前进行处理。

```
反例：构建除了一个虚拟的IncRate列
SELECT Income,Age,Income/Age as IncRate FROM datasets.hits_v1;
正例：拿到 Income 和 Age 后，考虑在前端进行处理，或者在表中构造实际字段进行额外存储
SELECT Income,Age FROM datasets.hits_v1;

```

## 8、用IN代替JOIN

clickhouse支持使用JOIN进行关联查询，但是他的实现机制是将后面的表全部加载到内存中，然后跟前表数据在内存中进行合并。

当表的数据比较大时，对内存的消耗是非常大的。所以通常情况下，可以用IN代替JOIN。如果非要用JOIN时，也要尽量把大表写在前面，小表写在后面。这跟mysql建议用小表驱动大表恰好相反。

例如下面两个语句的查询逻辑基本上是一样的。但是查询耗时差距却非常大。

```

select a.CounterID from datasets.hits_v1 a where a.CounterID in (select CounterID from datasets.visits_v1) limit 100;
-- 结果 100 rows in set. Elapsed: 0.047 sec. Processed 262.02 thousand rows, 1.05 MB (5.55 million rows/s., 22.21 MB/s.)

select a.CounterID from datasets.hits_v1 a left join datasets.visits_v1 b on a. CounterID=b.CounterID limit 100;
--结果 100 rows in set. Elapsed: 0.156 sec. Processed 1.68 million rows, 6.71 MB (10.74 million rows/s., 42.98 MB/s.)
```

# 七、生产常见问题

## 1、Clickhouse的数据一致性问题

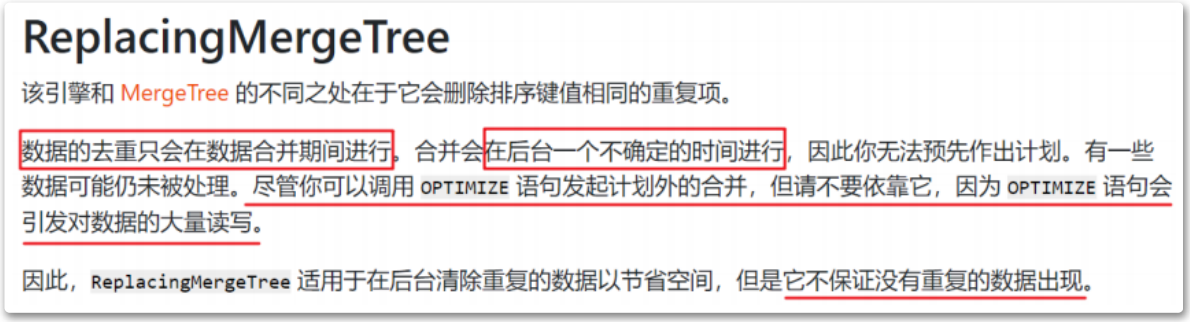
在生产环境中，数据一致性的重要性，不论如何强调都不过分。而clickhouse在进行数据变更时，都会产生一个临时分区，而不会更改原始数据文件，对数据文件的修改操作会要等到数据合并时才进行。所以clickhouse只能保证数据的最终一致性，而不能保证强一致性。很可能数据变更后，程序通过clickhouse查到之前的错误数据。因此使用clickhouse，要尽量比较数据的增删改这类数据变更操作。但是实际使用时，又不可避免的要使用数据变更操作。这时就需要有一套策略来全面处理数据一致性问题。

首先，对于分布式表，最好的办法是尽量避免使用。如果非要使用分布式表，一定要打开internal_replication。每个分片一定要配置多副本机制，使用副本机制来保证副本之间的数据一致性。

一般来说，分布式表会带来非常多的问题。往分布式表中导入数据时，数据是异步写入到不同的分片当中的，这样数据写入过程中就不可避免的有先有后。在最后一个分片的数据写入完成之前，不可避免的就会产生数据一致性的问题。

另外，对于分布式表，如果在数据写入时，这个分片的服务宕机了，那么插入的数据就有可能会丢失。clickhouse的做法是将这个数据分片转移到broken子目录中，并不再使用这个数据分片。也就是说，这时，clickhouse这一次的数据写入操作ius丢失了。造成的结果就是有可能就是一次update操作要更新1000条数据，但是最终却只更新了900条。

然后，对于本地的数据库，也一定要注意多副本造成的数据一致性问题。clickhouse中，即使是提供了去重功能的ReplacingMergeTree，他只能保证在数据合并时会去重，只能保证数据的最终一致性，而不能保证强一致性。



ReplacingMergeTree

该引擎和 MergeTree 的不同之处在于它会删除排序键值相同的重复项。

数据的去重只会在数据合并期间进行。合并会在后台一个不确定的时间进行，因此你无法预先作出计划。有一些数据可能仍未被处理。尽管你可以调用 `OPTIMIZE` 语句发起计划外的合并，但请不要依靠它，因为 `OPTIMIZE` 语句会引发对数据的大量读写。

因此，`ReplacingMergeTree` 适用于在后台清除重复的数据以节省空间，但是它不保证没有重复的数据出现。

对于MergeTree系列引擎，要注意他的合并操作不是定时的，是后台定时任务去自动进行merge合并操作。这个任务执行时间是无法设置或者掌控的。一般merge时间是在写入操作完成后的10~15分钟。但是如果某个分区一直不写入新的数据，那也有可能这个分区一直不会merge。这个时候，也只能通过optimize语句手动进行更新。

但是optimize语句强制合并数据CPU重操作，数据量大时，会非常消耗CPU资源，影响到线上的查询功能。因此，建议在晚上系统负载比较小的时候执行。另外，merge合并操作是没有锁的概念的，合并过程中依然可以正常写入。

实际生产中，在某些对数据一致性要求比较高的场景，可以自行采用乐观锁来屏蔽数据一致性的问题。例如，在创建一张表时，增加两个字段 sign和 version。sign表示这条数据是否删除，version数据表示这条数据的更新版本。像这样：

```
create table  A
(
xxx,
_sign UInt8,
_version UInt32
)
```

当进行数据更新时，不再进行更新操作，改为插入一条新的数据，同时version版本号加1。这样查询时，只要过滤verion版本号最大的一条数据就可以查询到最新的数据。

对于删除操作，同样改为新插入一条数据，version版本号加1的同时，把sign设置为-1，表示已删除。查询时，同样是找到版本号最大的这条数据，通过判断sign是不是等于-1，就能判断出这条数据是否被删除了。

但是这种方案需要注意过期数据要另行定期删除。

# 2、多副本表，尽量固定写入的节点

为了保证服务的高可用，企业使用clickhouse基本都会使用多副本的复制表。在clickhouse的复制表中，多个副本的地位是平等的，并没有主从之分。理论上写入任何一个节点都是可以的。clickhouse的副本之间会通过复制数据的方式进行同步。

但是，考虑到副本之间复制数据有可能会失败，在实际使用中，还是建议固定一个写入节点，作为主节点。因为当clickhouse服务出问题了，导致某一批次数据写入操作失败时，clickhouse只能保证数据块的写入是原子性的，而并不保证整批数据是原子性的。这样当服务恢复过来后，就可能造成有些数据写成功了，有些数据写失败了。这时就需要将这些写入成功的数据块还原回去，才能对这一批次数据重新进行写入。

这时，可以从副本中对数据进行全量恢复。恢复的方法很简单，将主节点的metadata和data目录全部清空，然后将副本节点的metadata和data目录拷贝过来，然后重启数据库就可以了。

## 3、Zookeeper数据丢失导致副本表无法启动

clickhouse的副本表需要由zookeeper提供集群信息。这时，如果zookeeper服务出问题了，导致丢失了一部分副本表的表信息，而clickhouse的metadata元数据中依然存在zookeeper的信息，就会导致启动报错。

```
Can’t get data for node /clickhouse/tables/01-
02/xxxxx/xxxxxxx/replicas/xxx/metadata: node doesn’t exist (No node):
Cannot attach table xxxxxxx
```

这时就需要手动重建zookeeper里的信息。手动恢复当然不太可能。这时可以移除问题节点上的metadata中对应表的结构文件，将clickhouse服务启动起来。启动完成后，重新创建表即可。建表的语句可以从metadata中的表结构文件中获取，也可以从其他正常节点上执行 show create table 语句获取。

而clickhouse中对应的表数据，会在表重建后重新下载。这样过一段时间再来验证一下数据是否一致就可以了。

最后，关于clickhouse常见的一些问题，可以参考下阿里云中提供的常见问题列表。https://help.aliyun.com/document_detail/162815.html。

# 八 clickhouse总结

整体来看，clickhouse以一己之力，就足够支撑一个完整的数仓功能。这与hadoop体系需要非常多的组件通力合作，形成了鲜明的对比。

clickhouse极大的挖掘了服务器的性能。强大的数据写入性能、极其高效的查询性能、高效的压缩存储以及大数据查询的强大吞吐量 ，这些特点使得clickhouse即使单机部署，也丝毫不逊色于传统的大数据集群。而他的集群功能也只需要通过zookeeper将单机节点松散的组合到一起，这使得他的水平扩展比常见的一些大数据平台更为简单高效。

在clickhouse极强性能的背后，他的使用体验相比于其他大数据产品，却显得极为简单直接。所有功能都是从几个极少的扩展点有序的堆叠扩展。clickhouse的使用体验也是非常舒适自然。使用体验简单的背后，其实也代表着运维的工作相当轻松。相比Hadoop体系以及其他大数据产品各种各样的参数调优，版本兼容，clickhouse的运维工作就显得简单多了。

另外，最重要的是，clickhouse的版本更迭相当的迅速。BUG修复速度非常快，并且新功能也层出不穷。目前体现出来的很多实验阶段的特性都非常有吸引力。而clickhouse主动兼容jdbc、mysql和postgresql这些成熟产品，也使得他的周边生态非常成熟。使用clickhouse基本没有什么技术门槛，当然，要用好还是没那么简单的。

综合这些显著的特点，clickhouse非常适合用来搭建数据仓库。未来进行数仓产品选型时，clickhouse是一个不得不考虑比较的重要产品。