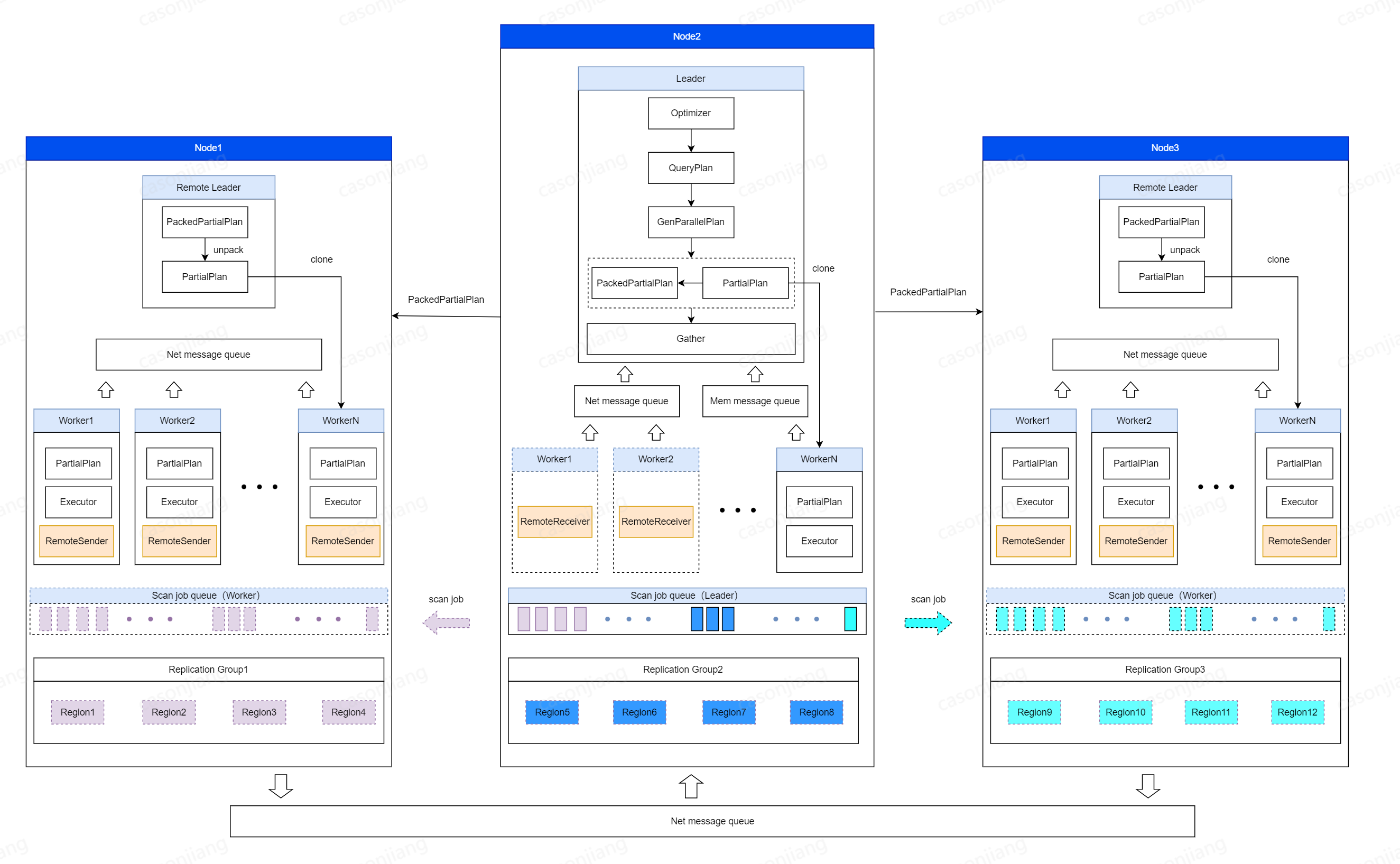
# 背景

# 概念

# 方案

参考ParallelQuery中对并行所依赖的数据结构做的clone操作，将需要clone的内容通过序列化/反序列化的方式支持多机执行。



## ParallelScan全局划分

leader做总的Parallen scan job划分，其他worker会通过从leader scan job queue获取scan job执行。

1、数据亲和性的任务调度

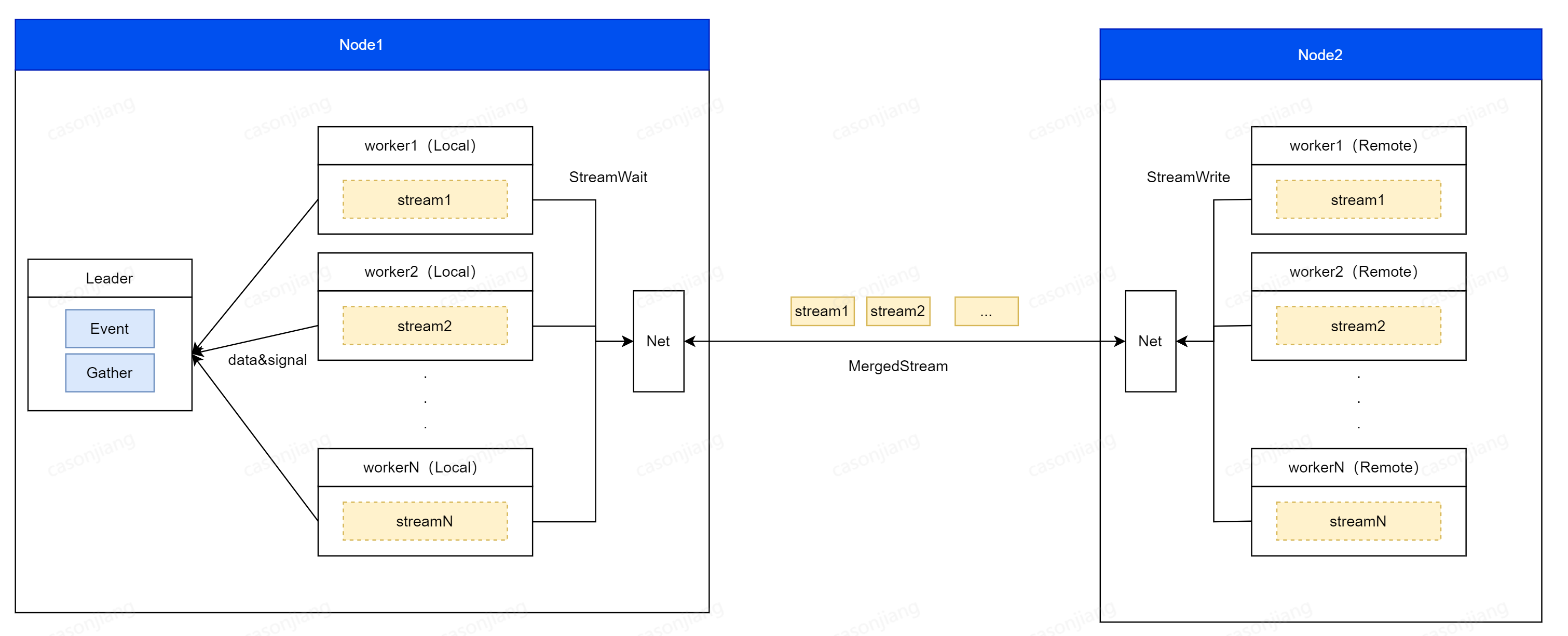
leader node在分配task给其他worker时应尽可能考虑数据亲和性，例如sql1 parallel scan涉及到的数据主要在node1和node2两个节点上，那么leader在分发task的时候应该尽量分配给node1和node2而不是node3，这样也可以达到类似存算分离架构下的“下推”效果。

每个node上的worker会优先向leader请求数据分布在当前node上的scan job。

2、read view跨节点传递

3.0的read view本质上就是一个时间戳，只要把这个时间戳传递给其他node，并且其他node用这个时间戳构建自己的transaction snapshot就可以。

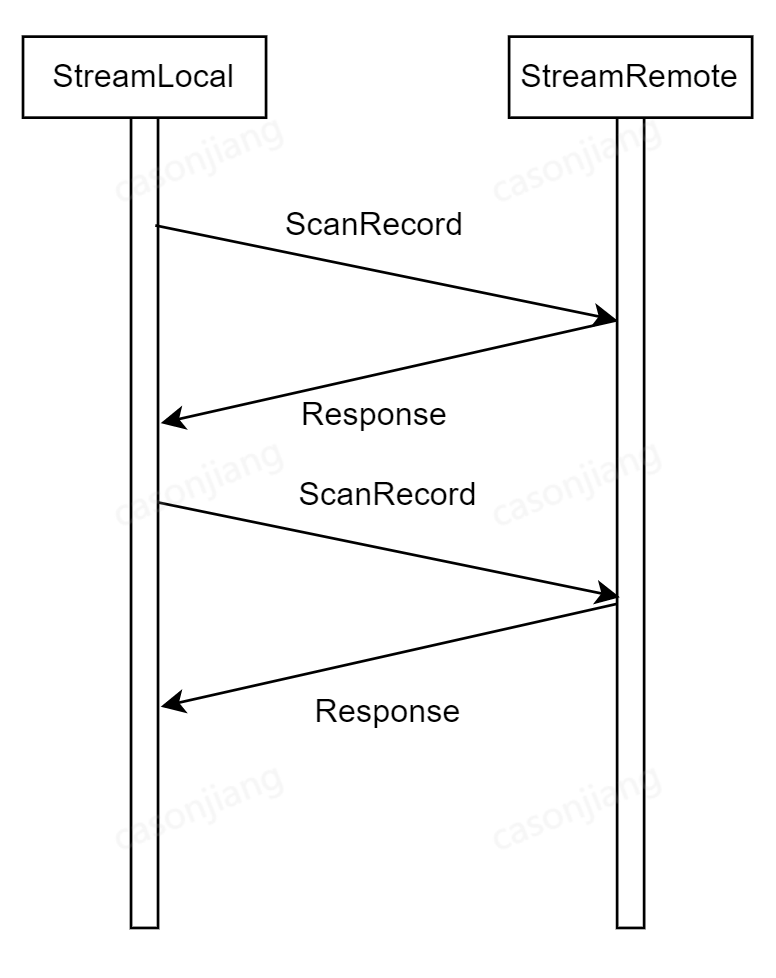
## Leader和Worker之间的NetMessageQueue



1、Leader使用no wait的方式逐个Stream做Try Read，有数据则读取，都没有数据则Wait在Event上

2、每个worker（Local）通过StreamWait发现流上有数据会通过Leader Event 唤醒Leader

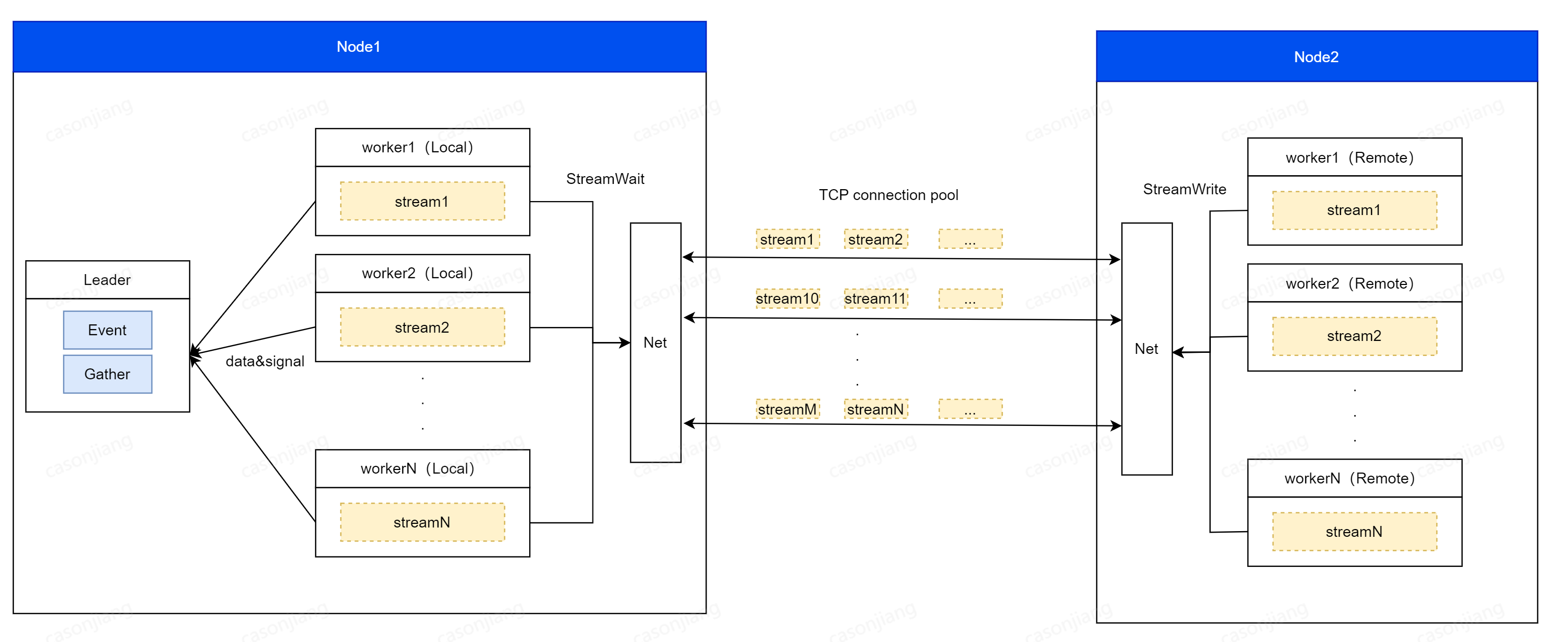
实现规划：一期



先封装NextStream rpc请求模拟Stream形式，存在的问题是多次RPC之间存在rtt等待间隔，如果RPC次数过多等待延迟消耗过大

但是考虑到一期主要针对一阶段并行，worker返回的数据量一般不会很大，rtt次数不会很多也可以接受

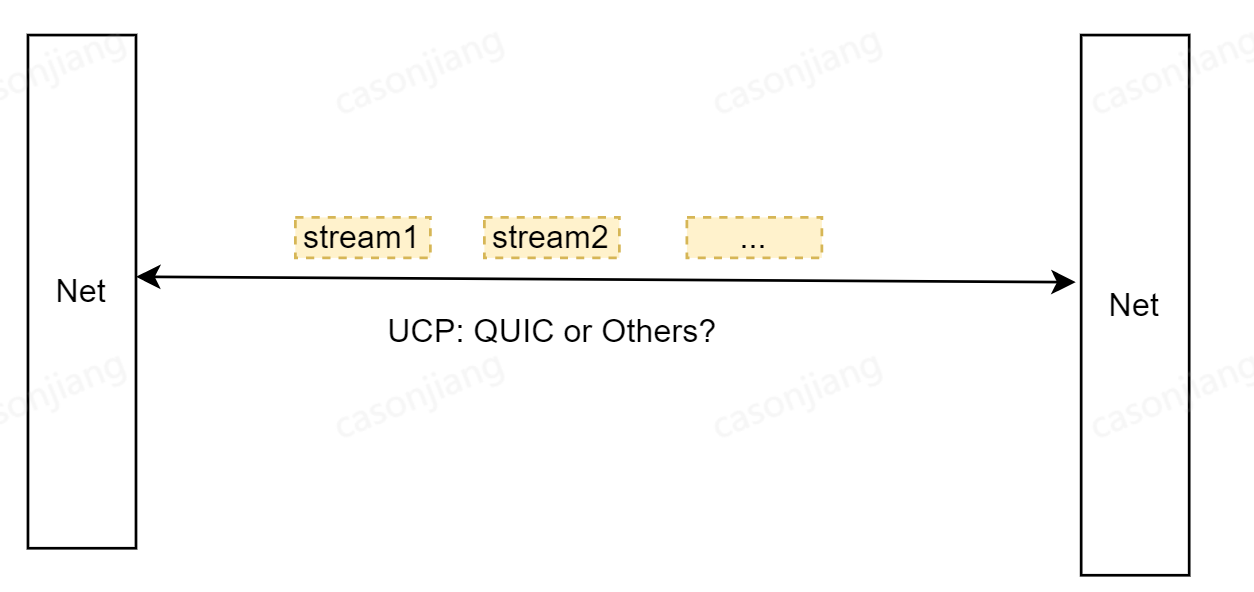
实现规划：二期



考虑到后续实现多阶段并行的MPP时，比如shuffle，broadcast等算子操作需要worker之间也需要通过网络进行数据交互，此时再使用一期的rpc机制就会带来rtt过多网络等待时间占比过高的问题，需要实现更高效的stream形式的传输机制

由于tcp存在队头阻塞问题，上层构建的多个stream会互相影响，本质上是因为上层构建的多个stream到tcp层都会共用同一个tcp连接基于滑动窗口的拥塞控制，这可以先通过建立tcp connection pool的形式来缓解，但不能根治。

实现规划：三期



如果二期的实现在实测后发现tcp队头阻塞问题的影响还是比较大将会进一步优化为在udp基础上做的可靠传输的stream protocol（可参考QUIC或公司其他团队成果），让每个stream拥有独立的拥塞控制，避免tcp队头阻塞带来的多个stream互相影响问题。

## Leader对worker的管理

如果leader挂了，那么各个worker也需要通过心跳机制在感知到leader down掉后及时终止释放计算资源。

如果用户kill掉了某个query，那么leader也需要通知各个worker及时终止释放计算资源。

## QueryPlan序列化/反序列化

### 背景

之前条件下推的时候，使用了逆波兰表达式，比较简单和直观，可以满足大部分的情况，但是由于mysql内部的各种转换，一些极端情况会有问题，如：

https://git.woa.com/tdsql3.0/SQLEngine/issues/869

序列化和反序列化可以精准的表示还原出需要传递的内容，并且天然支持跨机，可以支持更加复杂细粒度的内容，如后续的执行计划片段。

### Item序列化

目前不支持的Item

1）并行本身不支持

Item\_sp\_variable:Item\_splocal,Item\_case\_expr

Item\_name\_const

Item\_ident\_for\_show

Item\_outer\_ref

Item\_ref\_null\_helper

Item\_insert\_value

Item\_trigger\_field

Item\_udf\_func

Item\_func\_set\_user\_var

Item\_user\_var\_as\_out\_param

Item\_func\_match

Item\_func\_sp

Item\_subselect

Item\_udf\_sum

2）序列化未实现

Item\_geometry\_func

Item\_func\_array\_cast

Item\_metadata\_copy

Item\_cache\_row

Item\_aggregate\_type:Item\_type\_holder,Item\_values\_column

Item\_func\_json\_overlaps

Item\_func\_group\_concat

Item\_sum\_collect

Item\_rollup\_sum\_switcher

Item\_func\_grouping

Item\_nth\_value

Item\_first\_last\_value

Item\_non\_framing\_wf

### 条件下推

目前通过序列化和反序列化已经支持常用的Item和mysql内部的一些数据结构，基于序列化已经重构完成之前的所有条件下推逻辑，并且解决上述的问题

1）基于开源库实现基本数据类型的序列化/反序列化

2）序列化和反序列化使用一个函数，保证一致性

3）Item结构的序列化

思路是：通过脚本为每个要序列化的Item类生成class id的宏，和一个object factory函数为每个序列化的类调用默认构造函数，来解决多态下的对象生成问题。每个要序列化的Item要实现：

serialize()函数

默认构造函数

class\_id()类函数的声明（脚本来生成class\_id的实现）

以上实现的好处是类对象和序列化的标识维护对应关系比较robot，容易维护，在升级了mysql的基线版本后，再运行一下脚本做一下修正就行。

TODO: 脚本支持Update模式：脚本支持读取目标已经生成好的源文件，不修改里面已经存在的entry，只为新的Item类加entry，这样保持已有的class id不变，保证序列化协议的兼容。

4）String,DTCollation，in\_vector，cmp\_item,List等内部数据结构的序列化/反序列化

具体代码：https://git.woa.com/tdsql3.0/SQLEngine/commit/c7dc99735b913d30610806f26a3fbddeab74e51c

### QueryPlan序列化

和条件下推类似，只是需要对更多的结构进行序列化/反序列化，基于并行的架构并行查询，这里只需关注access\_path结构的序列化，不需要关注如何拆分计划

1）THD

2）query\_expression，query\_block

3）fields，table\_list

4）JOIN

5）filesort

5）access\_path

并行查询已经实现了计划的重写，以及子任务基于计划后如何执行，因此这里更多的只是把原先内存拷贝的操作转成序列化和反序列化的逻辑

### SessionVariable

整理一下和执行计划相关的session variable，只序列化这些

ulonglong max\_heap\_table\_size;

ulonglong tmp\_table\_size;

bool end\_markers\_in\_json;

bool windowing\_use\_high\_precision;

/\* A bitmap for switching optimizations on/off \*/

ulonglong optimizer\_switch;

ulonglong optimizer\_trace; ///< bitmap to tune optimizer tracing

ulonglong optimizer\_trace\_features; ///< bitmap to select features to trace

long optimizer\_trace\_offset;

long optimizer\_trace\_limit;

ulong optimizer\_trace\_max\_mem\_size;

sql\_mode\_t sql\_mode; ///< which non-standard SQL behaviour should be enabled

ulonglong option\_bits; ///< OPTION\_xxx constants, e.g. OPTION\_PROFILING

ha\_rows select\_limit;

ha\_rows max\_join\_size;

uint cte\_max\_recursion\_depth;

ulonglong histogram\_generation\_max\_mem\_size;

ulong join\_buff\_size;

ulong lock\_wait\_timeout;

ulong max\_sort\_length;

ulong min\_examined\_row\_limit;

ulong optimizer\_prune\_level;

ulong optimizer\_search\_depth;

ulonglong parser\_max\_mem\_size;

ulong range\_optimizer\_max\_mem\_size;

ulong preload\_buff\_size;

ulong profiling\_history\_size;

ulong read\_buff\_size;

ulong read\_rnd\_buff\_size;

ulong div\_precincrement;

ulong sortbuff\_size;

ulong max\_sp\_recursion\_depth;

ulong default\_week\_format;

ulong max\_seeks\_for\_key;

ulong range\_alloc\_block\_size;

ulong query\_alloc\_block\_size;

ulong query\_prealloc\_size;

ulong trans\_alloc\_block\_size;

ulong trans\_prealloc\_size;

ulong group\_concat\_max\_len;

ulong rbr\_exec\_mode\_options; // see enum\_rbr\_exec\_mode

ulong completion\_type;

ulong transaction\_isolation;

ulong updatable\_views\_with\_limit;

​

/\* Only charset part of these variables is sensible \*/

const CHARSET\_INFO \*character\_set\_filesystem;

const CHARSET\_INFO \*character\_set\_client;

const CHARSET\_INFO \*character\_set\_results;

​

/\* Both charset and collation parts of these variables are important \*/

const CHARSET\_INFO \*collation\_server;

const CHARSET\_INFO \*collation\_database;

const CHARSET\_INFO \*collation\_connection;

​

Time\_zone \*time\_zone;

/\*

TIMESTAMP fields are by default created with DEFAULT clauses

implicitly without users request. This flag when set, disables

implicit default values and expect users to provide explicit

default clause. i.e., when set columns are defined as NULL,

instead of NOT NULL by default.

\*/

bool explicit\_defaults\_for\_timestamp;

​

bool sysdate\_is\_now;

​

char \*track\_sysvars\_ptr;

bool session\_track\_schema;

bool session\_track\_state\_change;

ulong session\_track\_transaction\_info;

​

ulong internal\_tmp\_mem\_storage\_engine; // enum\_internal\_tmp\_mem\_storage\_engine

​

const CHARSET\_INFO \*default\_collation\_for\_utf8mb4;

​

## Open table多版本和Online DDL

leader open table后会在PartialPlan中加入table的schema\_version，在remote worker上open table的时候如果发现对应table的schema version和leader传下来的不匹配则报错，后续需要ddl流程做进一步优化来减少这里的报错。

# 开发规划

1、第一阶段

对于目前的并行还不支持的执行计划会回退到现在的下推实现上，完善目前Filter下推需要的Item field，Item compare function，Item int，Item string序列化。

2、第二阶段

1）实现多机ParallelScan机制，每个Worker只会从Leader获取数据在自己节点上的scan data job以覆盖下推功能

2）基于RPC形式实现NetMessageQueue

3）实现针对table scan，index scan，filter的MPP执行，需要序列化的结构有

QueryBlock & QueryExpression

JOIN

SessionVariable

table scan，index scan，Filter AccessPath

复用第一阶段的Item序列化

4）worker open table需要做schema version check，和leader不匹配则报错回退到单机执行

5）Leader对Worker之间的资源管理，包括心跳机制，terminal机制