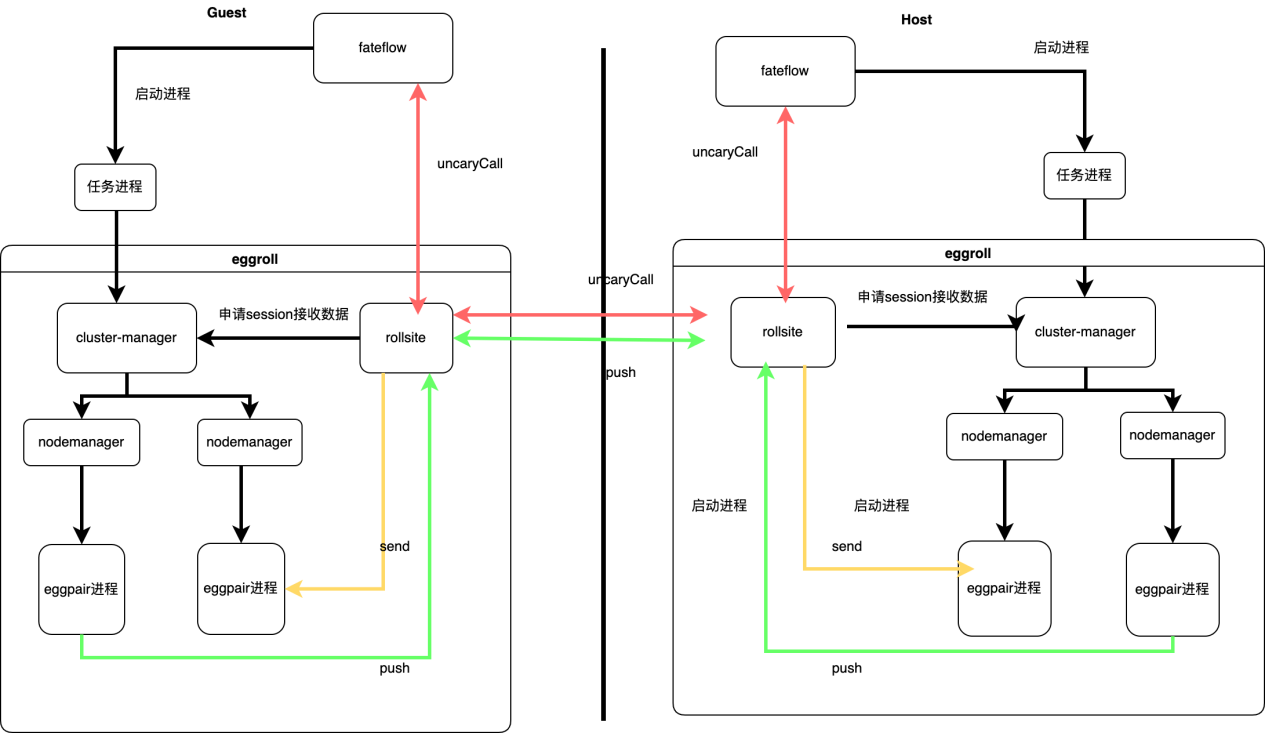
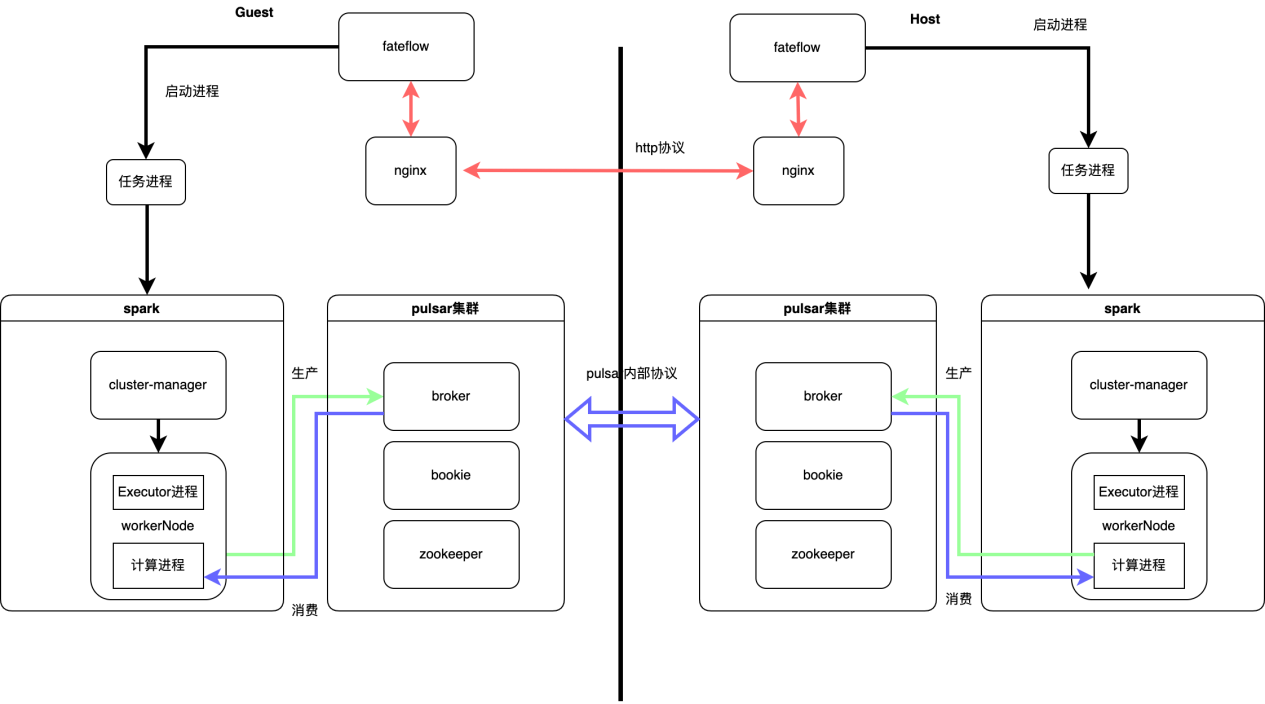
## 背景：

FATE1.X维护了多套通信架构，包括eggroll、spark+pulsar+nginx 、spark+rabbitmq+nginx。

下图为FATE1.X采用eggroll为计算引擎时的通信架构



下图为FATE1.X采用spark为计算引擎时的通信架构



如上所示，FATE1.X通信架构有一些弊端如下：

* 需要维护多套通信组件来支持不同的计算引擎
* 通信协议不统一，多种计算引擎组网困难，难以统一路由以及流控
* eggroll通信只支持同步rpc调用+流式传输，不支持消息队列类型的异步传输
* pulsar 以及rabbitmq等集群不易安装以及维护，程序不容易感知集群间传输时出现的网络异常
* 不容易对消息队列组件进行自定义开发等

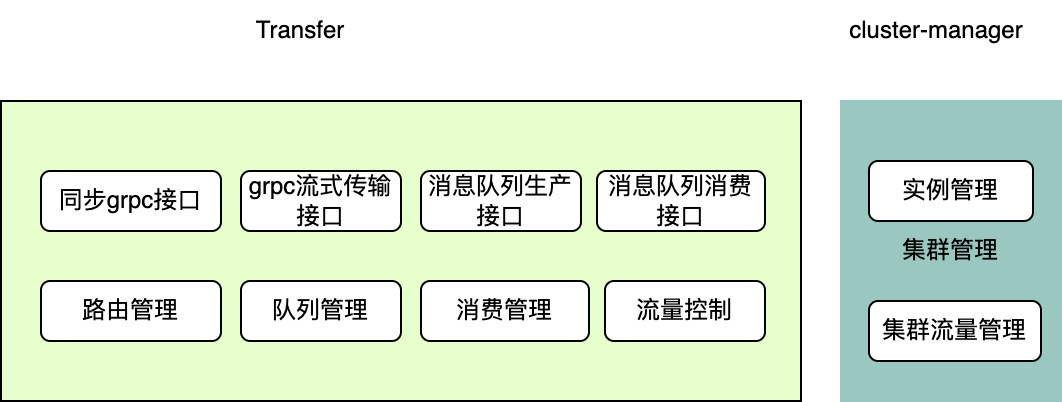
为了解决以上问题，我们预备在FATE2.X中使用统一的通信组件OSX，统一支持不同计算引擎

## 新组件OSX：

## 新组件特性：

* 通信协议统一使用grpc
* 支持多种计算引擎传输
* 同时支持同步rpc调用+消息队列
* 支持作为exchange中心节点部署 ，接入方在使用FATE1.X并将eggroll作为计算引擎时可以无缝接入
* 支持集群流量控制，可以针对不同参与方制定流控策略
* 路由配置与原eggroll一致，降低移植难度
* 支持集群模式与standalone两种模式（默认为standalone模式，standalone已可满足大部分场景）

## 组件设计：

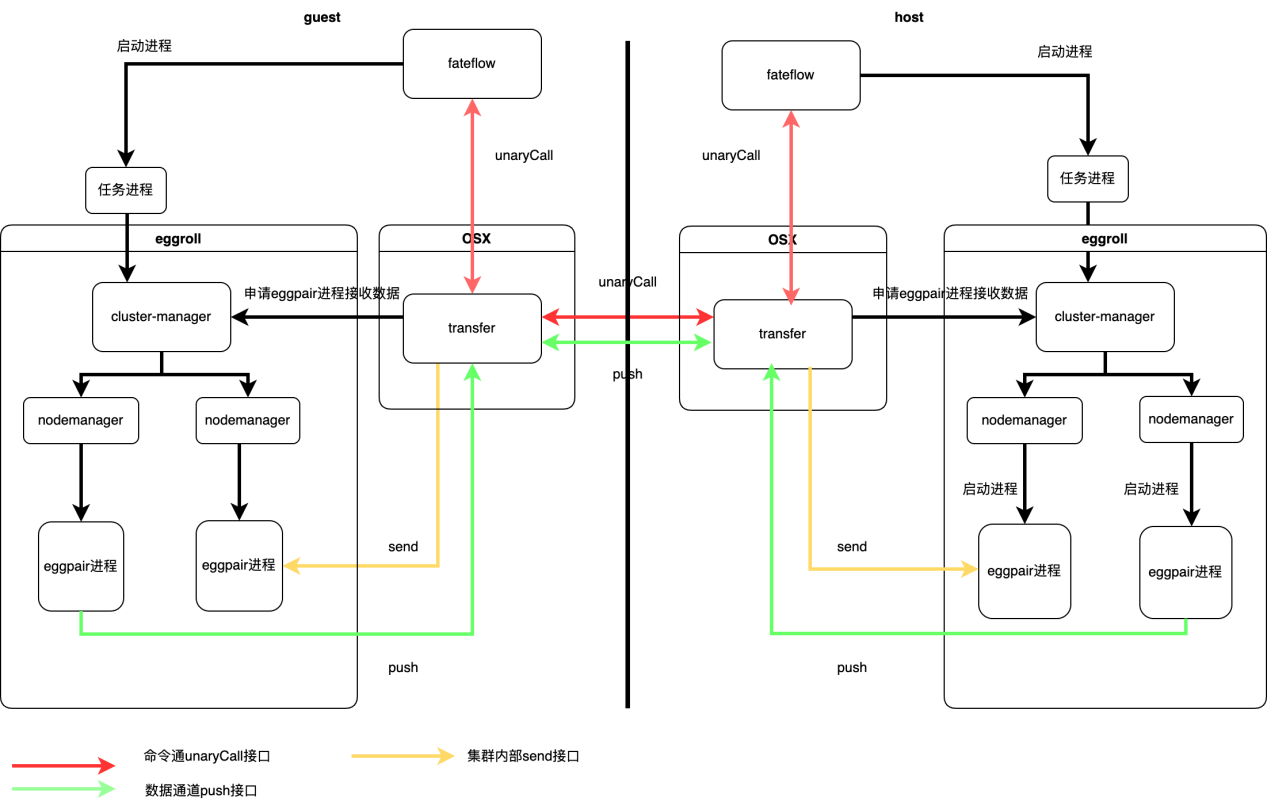


如上图所示，组件包括transfer 以及 cluster-manager。

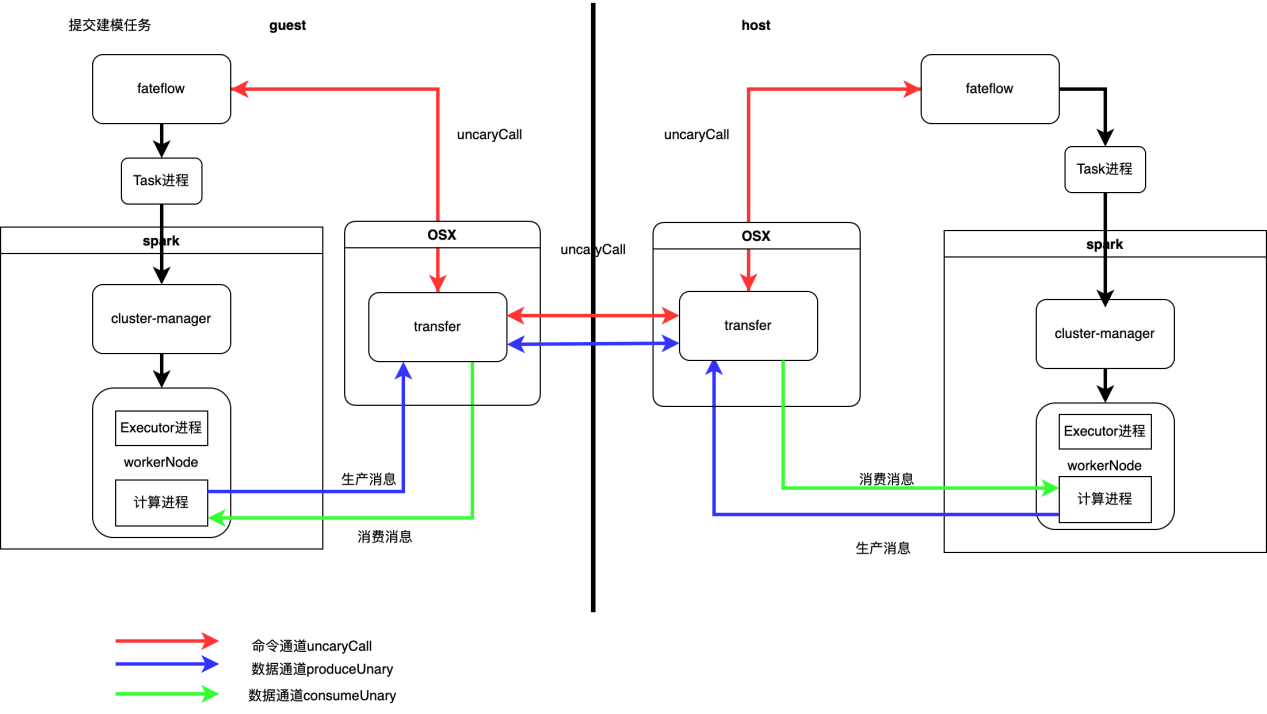
transfer：负责传输的主要组件，在standalone模式下只需要启动Transfer即 可。

cluster-manager（可选）: 在集群模式下，用于集群管理的组件，为进一步简化部署用户可以通过配置为zookeeper代替该实例。

## 部署架构：



上图为采用eggroll作为计算引擎时的部署架构



上图为采用spark作为计算引擎时的部署架构

## 详细设计：

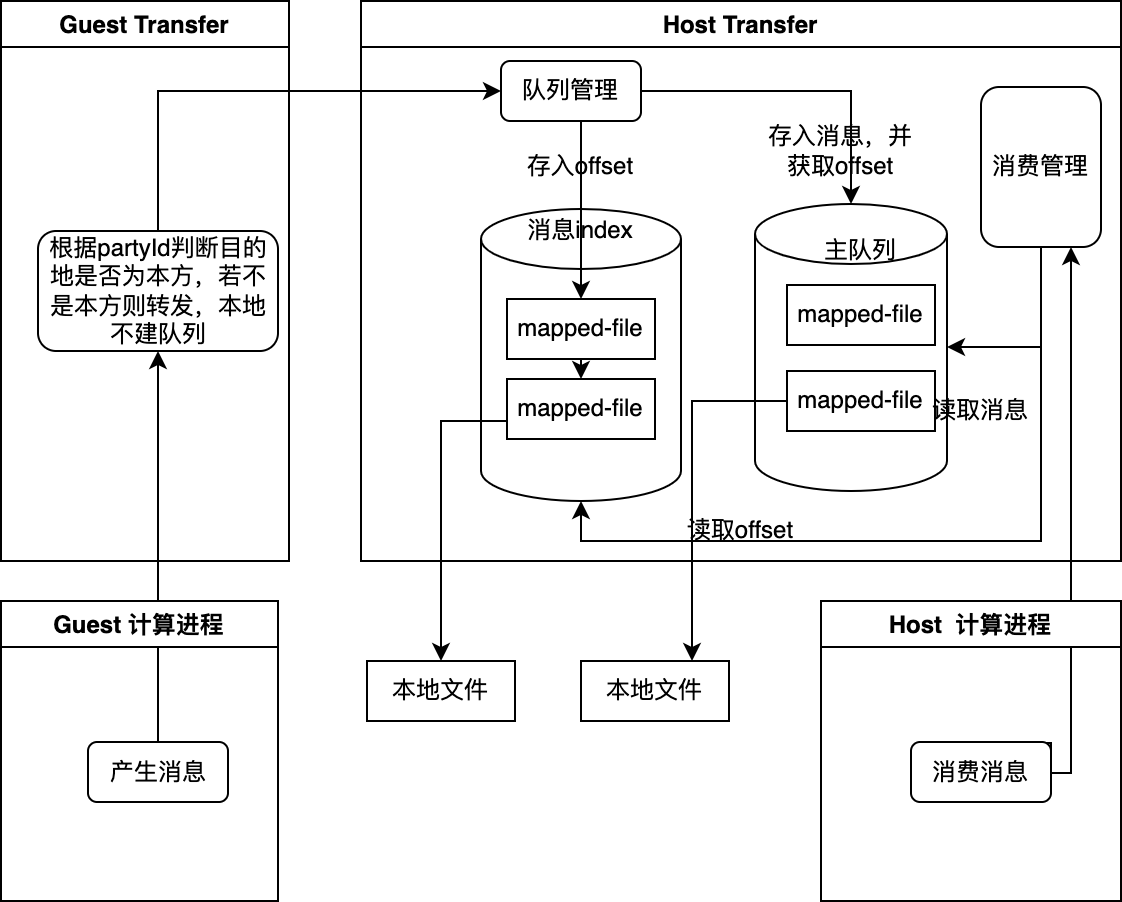
transfer采用了与eggroll的rollsite组件相同的grpc 同步调用接口，以及流式传输接口，所以能无缝地支持FATE1.X 以及FATE2.X使用eggroll作为计算引擎。

transfer组件对比eggroll 中的rollsite组件最大的改进是新增了消息队列功能。

用于代替pulsar以及rabbitmq。

###### 消息传递以及存储

下图为Transfer消息传递逻辑以及存储结构。



主存储队列：由一个经过mmap的文件列表组成，每个文件默认大小为1G，当文件写满会自动创建下一个文件。

indexFile：由一个经过mmap的文件列表组成，每个文件默认大小为64M，用于存储消息在主队列中的offset，文件写满会自动创建下一个文件。

存取消息都需要经过两次队列操作，由于使用mmap，使得存取速度较快。

###### 消息消费：

支持两种模式：

1. 服务端维护消费进度，并在无消息到来时提供阻塞功能。
2. 客户端自行维护消费进度，可以从队列的任意位置进行消费。

###### 路由：

OSX兼容eggroll的传输接口，路由表的配置与eggroll路由表配置一致，减少学习以及迁移成本。

{  
 "route\_table":  
 {  
 "9999":  
 {  
 "default":[  
 {  
 "port": 9370,  
 "ip": "192.168.0.1"  
 }  
 ]  
 },  
 "10000":{  
 "fateflow":[  
 {  
 "port": 9360,  
 "ip": "localhost"  
 }  
 ]  
 }  
 },  
 "permission":  
 {  
 "default\_allow": true  
 }  
}

* **流量控制：**

OSX提供了流控功能（默认为关闭），可对不同partyId的上行下行流量分别进行控制。

在开启流控功能的情况下，实现方式为通过使用滑动窗口对某一方向流动的数据进行带宽统计，新数据到来之后先从滑动窗口中判断是否超过带宽限制，若已超过则让传输线程休眠一段时间。

* **安全：**

OSX可通过配置证书的方式使用Https传输，该方式与FATE1.X 一致。

* **Polling模式：**

在实际部署场景中，会出现联邦学习的某些参与方由于策略原因不能对外提供端口用于rpc传输， OSX可配置使用Polling模式适配该场景。

###### 接口：

OSX提供的所有的接口都使用grpc协议，协议内容使用protobuf表示，具体可以参照源码中proto文件。

produceUnary 消息生产者接口，该接口用于使用队列传输模式的消息生产者调用,该接口可根据消息中的路由信息

rpc produceUnary( ProduceRequest) returns (ProduceResponse);

请求参数：

message Message{  
 string msgId = 1;//消息ID  
 bytes body = 2;//消息体  
 bytes head = 3;//消息头部  
}  
message RouteInfo{  
 string srcPartyId =1;//源partyId  
 string srcRole = 2;//源角色  
 string desPartyId = 3;//目的partyId  
 string desRole = 4;//目的角色  
}

返回内容：

message ProduceResponse {  
 string transferId = 1;//唯一代表一个传输队列，等同于topic  
 string sessionId = 2;//唯一代表一个传输会话  
 int32 code = 3;//返回码  
 string msg = 4;//返回信息  
}

消费接口：consumeUnary该接口用于队列模式的消费者调用，支持服务端以及客户端维护消费进度两种模式

rpc consumeUnary (ConsumeRequest) returns (ConsumeResponse);

请求参数：

message ConsumeRequest {  
 string transferId = 1;//唯一代表一个传输队列，等同于topic  
 string sessionId = 2;//唯一代表一个传输会话  
 int64 startOffset =3;//消息index ，传-1则使用服务端维护的消费进度  
}

返回数据：

message ConsumeResponse {  
 int64 startOffset = 1;//消息index  
 int64 totalOffset = 2;//当前队列总长度  
 Message message = 3;//消息内容  
 int32 code = 4;//返回码  
 string msg = 5;//返回消息  
 string transferId = 6;//唯一代表一个传输队列，等同于topic  
}

消费Ack接口

该接口用于消费者消费队列消息时确认已消费。若客户端自行维护消费进度，则不需要调用该接口。

rpc ack (AckRequest) returns (AckResponse);

message AckRequest{  
 string transferId = 1;//唯一代表一个传输队列，等同于topic  
 string sessionId = 2;//唯一代表一个传输会话  
 int64 startOffset =3;//消息的index  
}  
  
message AckResponse{  
 int32 code = 1;//返回码 0位正常  
 string msg = 2;// 返回消息  
}

查询传输队列接口：

该接口用于在集群模式下，查询某个传输队列所在的实例

rpc queryTransferQueueInfo(QueryTransferQueueInfoRequest) returns (QueryTransferQueueInfoResponse);

请求参数：

message QueryTransferQueueInfoRequest{  
 string transferId = 1;//唯一代表一个传输队列，等同于topic  
 string sessionId = 2;//唯一代表一个传输会话  
}

返回内容：

message QueryTransferQueueInfoResponse{  
 int32 code = 1;//返回码 0为正常  
 string msg = 2;//返回信息  
 repeated TransferQueueInfo transferQueueInfo = 3;//队列信息  
}  
message TransferQueueInfo{  
 string transferId = 1;//  
 string ip = 2;//队列所在实例ip  
 int32 port = 3;//队列所在实例端口  
 int64 createTimestamp = 4;//创建时间  
 int32 status = 5;//队列状态0 为正常  
}

unaryCall接口为同步grpc接口，继承自FATE1.X，用于命令通道同步调用

push接口接口为流式传输接口，继承自FATE1.X ，用于数据通道流式传输

rpc unaryCall (Packet) returns (Packet);

rpc push (stream Packet) returns (Metadata);

参数以及返回

message Packet {  
 Metadata header = 1; // packet header  
 Data body = 2; // packet body  
}

message Metadata {  
 Task task = 1; // task description  
 Topic src = 2; // source topic  
 Topic dst = 3; // destincation topic  
 Command command = 4; // task managing command (if any)  
 string operator = 5; // model operator  
 int64 seq = 6; // stream seq (reserved)  
 int64 ack = 7; // stream ack (reserved)  
 Conf conf = 8; // operation config  
 bytes ext = 9;  
 string version = 100;  
}

message Topic {  
 string name = 1;  
 string partyId = 2;  
 string role = 3;  
 Endpoint callback = 4; // implication of   
}

message Endpoint {  
 string ip = 1; // ip address  
 int32 port = 2; // port  
 string hostname = 3; // host name  
}