**分布式系统课程报告**

**第一部分 – Hyperledger Fabric 介绍**

1. **什么是 Hyperledger Fabric？**

Hyperledger Fabric 是 Hyperledger（即“超级账本”）中的一个区块链项目，它是一个分布式账本技术，用于在联盟链（即由多个参与者组成的区块链网络）上构建和运行分类帐应用程序。

其中，分布式账本，是区块链网络的和核心，记录着网络上发生的所有交易。所有节点共同维护完全一致的账本数据，区块以保存前一个区块Hash值的方式顺序相连，组成了链式数据结构，保证了数据的不可篡改。具有以下的特点来保证其权威性、不可篡改性和安全性：

* 交易记录按照时间顺序排列，只能向后追加而不能篡改或删除已发生的交易；
* 去中心化，由每个节点共同维护，且都有一份完整的交易记录，即副本；
* 加密处理、存储和传输，保证交易记录的安全性。

1. **Hyperledger Fabric 的网络结构**

其中有多种不停类型的网络节点，不同的节点具有不同身份并提供不同职能，节点类型有如下几种：

**一、Peer节点**

1. 记账节点： 所有的Peer节点都是记账节点，负责验证从排序服务接收到区块中的交易，维护账本的副本；

2. 背书节点： 部分Peer节点会执行交易并对结果签名背书，充当背书 (Endorsement) 节点。背书节点是动态的角色，只有在应用程序向它发起背书请求的时候才是背书节点，其他时候只是普通的记账节点，只负责验证交易并记账；

3. 主节点： 代表的是与排序节点通信的节点，负责从排序服务节点处获取最新的区块并在组织内部同步。

**二、Orderer节点**

1. 排序服务节点接收包含背书签名的交易，并进行排序、打包生成区块，广播给Peer节点，保证同一个链上的节点接收到相同的消息，并且有相同的逻辑顺序；

2. 排序服务可以支持多链的交易处理，实现了多通道的数据隔离，保证只有同一个链的peer才能访问链上的数据；

3. 排序服务可选择集中式（Solo）或分布式（Kafka）协议，其中Kafka集群可以实现崩溃故障容错（CFT）。

**三、CA节点**

1. CA节点是Fabric的证书颁发机构；

2. CA节点接收客户端的注册申请，返回注册密码用于用户登陆，以便获取身份证书；

3. 在区块链网络上所有的操作都会验证用户的身份。

**四、Client节点**

1. 客户端节点代表最终用户操作的实体，它必须连接到某一个Peer节点或者排序服务节点上与区块链网络进行通信；

2. 客户端节点向背书节点提交交易提案，收集到足够背书后，向排序服务广播交易，进行排序，生成区块。

1. **Hyperledger Fabric 所使用的共识协议**

Hyperledger Fabric 使用一种叫做共识机制的算法来确定交易的执行顺序和决策是否被批准。共识机制是一种在分布式系统中达成一致的方法，通常是用来决定交易是否应该被记录到分类帐中。然而，共识不仅仅是简单地同意交易顺序，而是通过在整个交易流程中的基本作用，从提案和认可到订购，验证和承诺。Hyperledger Fabric 所使用的共识机制采用了模块化、插件化的设计方法，可以由使用者自由调用。

**第一种，SOLO模式的共识算法：**

在Hyperledger Fabric中的solo模式的共识算法，是最简单的一种共识算法，通过中央服务器来向其他所有节点提供服务。它由一个为所有客户服务的单一节点组成，所以不需要“共识”，因为有一个中央权威机构。相应地没有高可用性或可扩展性。这使得独立开发和测试很理想，但不适合生产环境部署。

**第二种，RAFT共识算法：**

Raft是一个分布式崩溃故障容错共识算法，它可以保证在系统中部分节点出现非拜占庭故障的情况下，系统依然可以处理客户端的请求。

对于Raft算法而言，每个节点只能处于三个状态之一：

跟随状态：初始情况下，所有的节点都处于跟随状态，也就是都是跟随节点。一旦某个跟随节点没有正常通信，它就转换为候选状态，也就是成为一个候选节点。跟随节点的日志可以被主导节点重写。

候选状态：处于候选状态的节点会发起选举，如果它收到集群中大多数成员的投票认可，就转换为主导状态。

主导状态：处理客户端请求并确保所有的跟随节点具有相同的日志副本。主导节点不可以重写其自身的日志。

如果候选节点发现已经选出了主导节点，它就会退回到跟随状态。同样，如果主导节点发现另一个主导节点的任期值（任期是一个单调递增的整数值，用来标识主导节点的管理周期。每个任期都从选举开始，直到下一个任期之前）更高，它也会退回到跟随状态。

所采用的领导者选举算法如下：要开始一个选举，跟随节点会递增其当前任期值并转换到候选状态。该节点首先给自己投一票，然后同时向其他节点发送请求投票的消息（RequestVote RPC消息）。候选节点会保持在候选状态，直到以下事件发生：

* 该节点胜出选举
* 其他节点胜出选举
* 没有节点胜出选举

如果该节点收到大部分节点的投票认可，就可以胜出选举，那么该节点就转换到主导状态成为新的主导节点。注意：每个节点只能投一票。如果同时也有其他节点宣布自己是主导节点并有更高的任期值，那么任期值高的节点成为新的主导节点。如果多个候选节点的得票情况相同，那么没有胜出节点。

这是一般情况下的领导者选举，只要有超过一半的节点投支持票了，候选节点才会被选举为 领导者。当领导者节点出现故障而又重新加入时，选举类似于我们课上所学的Bully算法，在这里不再赘述。

**第三种，PBFT共识算法：**

PBFT，Practical Byzantine Fault Tolerant，即实用拜占庭容错算法，顾名思义，该算法是为了解决联盟链中的拜占庭故障。与我们在课上老师所教学、推导的相同，这里的拜占庭容错算法对节点数的要求为n≥3f+1，其中，n是系统中的总节点数，f是允许出现故障的节点数。也就是说，如果这个系统允许出现f个故障，那么这个系统必须有n个节点，才能解决故障。

PBFT 模型下，有一个节点会被当做主节点，而其他节点都是备份节点（副本节点）。系统内的所有节点都会相互通信，旨在实现节点能以少数服从多数的原则达成数据的共识。如果主节点出现明显的撒谎迹象，那么其他的节点可以联合起来更换主节点。

PBFT算法的主要执行过程如下：

* 客户端发送请求给主节点
* 主节点广播请求给其它节点，节点执行pbft算法的三阶段共识流程
* 节点处理完三阶段流程后，返回消息给客户端
* 客户端收到来自f+1个节点的相同消息后，代表共识已经正确完成

**第二部分 – 课堂教学内容概述**

在课程开始，首先介绍的是**为什么会产生分布式系统**：因为利于资源的共享，更强的容错抗灾能力等等。

其次，动机之后就是分布式系统的**概念介绍、特性特点、与计算机网络的区别、存在的固有矛盾**即CAP理论以及设计一个分布式系统所**需要解决的问题**。

对于一个分布式系统，需要考虑的问题有：节点同步、数据一致性、容错能力、节点间通信、安全性等。提出了这些问题后，接下来就要考虑的是如何逐一解决这些问题，这样才能够实现一个分布式系统。

接下来，首先是**节点间的通信问题**，如何确保接收方能够正确地收到消息，且发送方能知晓接收方正确收到。这类似于网络课上所学的握手规则，同时反复对对方的消息进行确认会导致无休止的通信，产生资源的浪费与效率低下的问题。分析的结果是无法保证100%的可靠通信，实际上多采用了两消息会话与消息标识加上超时机制。

下面是**分布式系统中的选举问题**。通常在[分布式](https://so.csdn.net/so/search?q=%E5%88%86%E5%B8%83%E5%BC%8F&spm=1001.2101.3001.7020)集群中，需要一个主节点来负责对其他节点的协调和管理，这个选择主节点的过程在分布式领域中叫作分布式选举。主节点的存在，可以保证集群中其他节点的有序运行，并保证每个集群节点中数据的一致性，不存在不同的情况。这也就要求主节点必须高可用且有备份，如果主节点故障了，可能会导致节点上的数据不一致，所以需要一个选举算法来选出主节点并保证主节点的可用性。之后还讨论了选举算法的消息复杂度问题和同步异步网络中的不同算法。

之后是**分布式系统中的互斥问题**。和操作系统中的互斥问题相似，在有多个节点去访问同一个临界资源时，为避免冲突和错误，引入互斥算法。分布式系统中的互斥算法主要分为集中式（需要中央服务器）、基于令牌环与非令牌环的算法。在后面我们所做的大型停车场的方案设计中所讨论的基本也是这样的互斥问题。

下面是**分布式系统的同步问题**。在一个分布式系统中存在多个节点，节点之间的物理时钟也许会存在差异，这就可能导致在计算机之间同步消息时，出现对消息发生时间有分歧的情况。所以需要同步算法来保证节点之间事件的排序。引入了逻辑时间戳与向量时间戳来标记不同事件发生的先后顺序。以及还有一致全局快照算法，和节点状态的恢复。

接下来是**分布式系统中的故障与容错以及存在故障节点的共识问题**。简单来说，共识问题就是分布式系统需要解决的一个核心问题：在一个可能发生机器宕机、网络异常、数据篡改的环境下，如何让分布式系统中的所有节点快速准确的对某个数据值达成一致，且不会破坏整个系统的一致性。课上所提到的一个经典抽象模型就是拜占庭将军问题，即所描述的场景是：除了存在故障行为，还存在恶意行为的一个场景。应该根据实际的系统场景选择合适的算法，如果能确定该环境中各节点是可信赖的，不存在篡改消息或者伪造消息等恶意行为，推荐使用非拜占庭容错算法；反之，推荐使用拜占庭容错算法。还讨论了拜占庭容错对节点数的要求：n>3f。

最后是**分布式事务的提交**。在[分布式](https://so.csdn.net/so/search?q=%E5%88%86%E5%B8%83%E5%BC%8F&spm=1001.2101.3001.7020)系统中，事务往往包含多个参与者的活动，单个参与者的活动是能够保证原子性的，而保证多个参与者之间原子性则需要通过两阶段提交或者三阶段提交算法来实现。

**第三部分 – 对课程的建议**

在进行课程中的一些需要小组合作的作业过程中，从个人角度认为一些同学在进行团队合作时，尤其是编程任务中，有一些吃力，最终导致效率低下，由个别同学承担小组合作中的大量工作。当然不是本课程的原因，而是同学们也许没有这样的经验或团队协作能力，而且本身的编程能力较低、代码规范意识较差，也希望老师能够稍加引导，虽然编程能力和团队协作不是本课程的教学任务，但也希望又更多同学能在老师的帮助下得到能力、意识的提升。