一、共识算法的核心问题

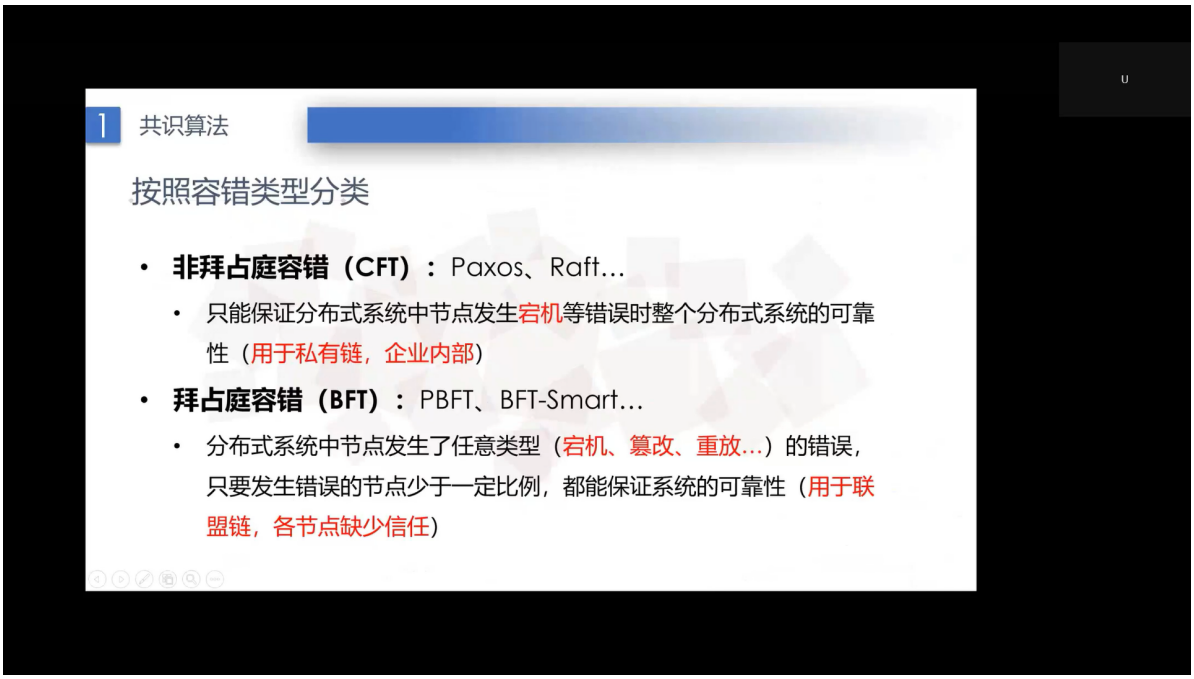
- 多个节点对某个数据达成一致共识
- 多个节点对多个数据的顺序达成一致共识(比如一个字符串，如果字符的顺序错了意思就错了)

> 注意：共识算法要求的是最终达成共识，而非过程中达成共识

二、区块链网络与共识算法关系

区块链网络中的节点产生的区块，当验证完其中交易的真实性之后不能直接传输到整个区块链网络中。还需要经过共识算法确认后才能完成上链。

三、共识算法的分类



> CFT : 仅在系统中节点发生宕机时保证整个分布式系统的可靠性,不能抵御恶意节点的攻击
>
> BFT : 系统中节点发生任意类型错误(包括恶意节点攻击)时，都可以保证整个系统的可靠性

四、PBFT算法

4.1 PBFT算法的目的

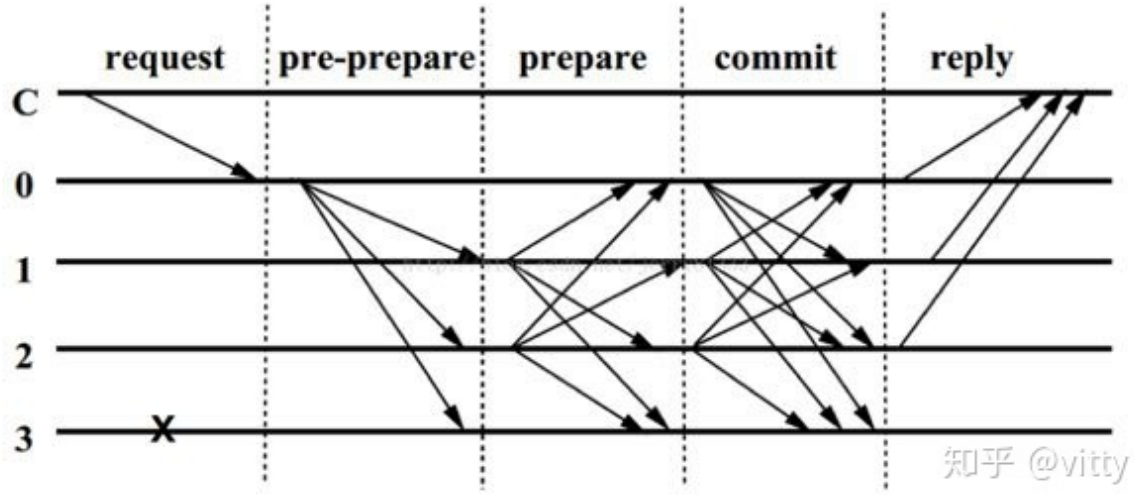
分布式系统中的好节点不知道哪些是坏节点，哪些是好节点。PBFT要做的就是让好节点达成共识。

4.2 PBFT共识过程

Boss向手下员工发送命令时，当员工认为此命令是有问题时会拒绝执行。即使认为这个命令是正确的，该员工也需要问团队中其他成功确定该命令是否是对的，只有当大多数人(2f+1)都认为Boss的命令是对的的时候，该员工才会去执行该命令。

4.3 PBFT共识流程展示

基于拜占庭将军问题，PBFT算法一致性的确保主要分为这三个阶段：预准备（pre-prepare）、准备(prepare)和确认(commit)。流程如下图所示：



我们首先解释一下上面各个符号表达的意思：

- C表示客户端；
- 0，1，2，3表示4个节点；
- 0在这里为主节点，1，2，3为从节点(也叫备份节点)；（注意，这里其他节点也可以作为主节点，若0发生错误只能由服务器监测。如果服务器在一段时间内不能完成客户端的请求，则会触发**视图更换协议**，将其他节点换为主节点）
- 3为故障节点；
- 视图（view）：一个view中存在一个主节点和多个从节点，它描述了一个多副本系统的当前状态。另外，节点是在同一个view上对数据达成共识，不能跨域view。

> 每个从节点的状态都包含了服务的整体状态，从节点上的消息日志(message log)包含了该从节点接受(accepted)的消息，并且使用一个整数表示从节点的当前视图编号（记作i）。

> **下面我们结合上图，详细说一下PBFT的步骤：**

- **Request**：请求端C发送请求到主节点，这里是0节点；

> 主节点的选举公式：p = v % |R| 。v是视图编号，p是主节点编号，|R|是副本集合的个数。

- **Pre-Prepare**：节点0收到C的请求后进行广播，扩散至1，2，3；
- **Prepare**：1，2，3节点收到后记录并再次广播，1->0/2/3，2->0/1/3，3因为宕机无法广播；（这一步是为了防止主节点给不同从节点发送不同的请求）
- **Commit**：0，1，2，3节点在Prepare阶段，若收到大于等于一定数量（2F+1，实际使用中，**F为可以容忍的拜占庭节点个数**）的相同请求，则进入Commit阶段，广播Commit请求；
- **Reply**：0，1，2，3节点在Commit阶段，若其中有一个收到大于等于一定数量（2F+1）的相同请求，则对C进行反馈；
- 客户端需要等待f+1个不同副本节点发回相同的结果，作为整个操作的最终结果。

> 根据上述流程，在 **N ≥ 3F + 1** 的情况下一致性是可能解决，**N为总计算机数，F为有问题的计算机总数。**

一个例子：

我们假设**N=4，F=1**，即有四个节点，其中有一个节点是坏的，我们还使用上面的图，即节点3为故障节点。

1. 请求端C发送请求到0节点，假设这里请求内容为“1”；
2. 节点0收到C的请求后进行广播，将请求内容“1”扩散至节点1，2，3；
3. 节点1、2、3收到后内容“1”后，再次广播，节点1->0，2，3，节点2->0，1，3，节点3因为宕机无法 广播；
4. 节点0，1，2会在上一阶段分别收到3个请求内容“1”(每个节点会给自身本地发送一个请求内容，同时在预准备阶段节点0发送给节点1,2,3的请求内容也会被包含)，均超过了2个，于是节点0、1、2会分 别广播请求内容“1”；
5. 此时如果一个节点（0，1，2中任意一个）收到3（2+1）条commit消息，即对C进行反馈。
6. 客户端C需要等待f+1个不同副本节点发回相同的结果，作为整个操作的最终结果。

4.4 为什么至少需要N=3F+1个节点以及PBFT限定条件

我们在上面讲到，当网络中有F台有问题的计算机时，至少需要3F+1台计算机才能保证一致性问题的解决，我们在这里讨论一下原因。

我们可以考虑：由于有F个节点为故障或被攻击的节点，故我们只能从N-F个节点中进行判断。但是由于异步传输，故当收到N-F个消息后，并不能确定后面是否有新的消息。（有可能是目前收到的N-F个节点的消息中存在被攻击的节点发来的消息，而好的节点的消息由于异步传输还没有被收到。）

我们考虑最坏的情况，收到的消息中有F条都是来自于坏节点，因此来自于好节点的消息是N -2F 条。我们必须要保证 N - 2F > F，即 N > 3F 。所以N的最小整数为N=3F+1。

> **PBFT算法有两个限定条件**

- 所有节点必须是确定性的，即：在给定状态和参数相同的情况下，操作执行的结果必须相同
- 所有节点必须从相同的状态开始执行

在这两个限定条件下，即使失效的从节点存在，PBFT算法对所有非失效从节点的请求命令总顺序达成一致，从而保证安全性。

五、PBFT 共识各阶段具体步骤

5.1 REQUEST

客户端C向主节点P发送<REQUEST, o, t, c>请求。o指请求的具体操作；t指请求时客户端追加的时间戳；c指客户端标识；REQUEST包含消息内容m，以及消息摘要d。客户端C发出请求的时间戳是顺序排列的，后续发出的请求比早先发出的请求拥有更高的时间戳。

主节点P收到客户端C的<REQUEST, o, t, c>请求，需要进行以下校验：

- 客户端请求REQUEST中消息签名是否正确(通过消息内容m和消息摘要d进行判断)。

如果验证不通过，则丢弃，否则接受消息，于是进入 PRE-PREPARE 阶段。

5.2 PRE-PREPARE

此阶段中，**主节点给收到的消息分配一个编号n**，接着广播一条<<PRE-PREPARE, v, n, d>，m>消息给其他副本节点，并将请求记录到本地历史（log）中。说明：**n主要用于对所有客户端的请求进行排序**；v指视图编号；m指消息内容；d指消息摘要。

从<PRE-PREPARE, v, n, d>可以看出，请求消息本身内容（m）是不包含在预准备的消息里面的，这样就能使预准备消息足够小。预准备消息的目的是作为一种证明，确定该请求是在视图v中被赋予了序号n，从而在**视图变更**的过程中可以追索。

副本节点收到主节点的<<PRE-PREPARE, v, n, d>, m>消息，需要进行以下校验：

- REQUEST和PRE-PREPARE消息中签名是否正确。
- 当前视图编号是v。
- 该节点从未在视图v中接受过序号为n但是摘要d不同的消息m(序号为n的消息m的摘要d只能有一种情况)。
- m经过计算后的消息摘要与消息中消息摘要d是否一致。
- 判断n是否在区间h, H内

5.3 PREPARE

此阶段中，当前**副本节点**广播一条<PREPARE, v, n, d, i>消息，并且将**预准备消息**和**准备消息**写入自己的消息日志。i是当前节点编号。

主节点和副本节点收到<PREPARE, v, n, d, i>消息，需要进行以下校验：

- 消息签名是否正确。
- 判断n是否在区间[h, H]内。
- 信息摘要d是否和当前已收到 PRE-PPREPARE 中的d相同(序号n相同的消息进行比较)

如果验证不通过，则丢弃，否则接受消息。PREPARE准备阶段完成的条件为：**当前节点i将(m,v,n,i)写入消息日志，从2f个不同副本节点收到的与预准备消息一致的准备消息。**满足这两个条件后进入COMMIT阶段。

5.4 COMMIT

此阶段中，当前节点广播一条<COMMIT, v, n, d, i>消息。

主节点和副本节点收到<COMMIT, v, n, d, i>消息，需要进行以下校验：

- COMMIT消息签名是否正确(是否有人伪造他人身份发送COMMIT消息)。
- 当前副本节点是否已经收到了同一视图v下的n。
- 计算m的摘要，并判断和d是否一致。
- n是否在区间[h, H]内。

如果验证不通过，则丢弃，否则接受消息。**COMMIT阶段完成的条件是**：

- 任意f+1个正常副本节点集合中prepared(m,v,n,i)为真，这一条确保committed(m,v,n)为真。
- prepared(m,v,n,i)为真，并且节点i已经接受了2f+1个确认（包括自身在内）与预准备一致的消息。这一条确保committed-local(m,v,n,i)为真。确认与预准备消息一致的条件是具有相同的视图编号、消息序号和消息摘要。

> 完成COMMIT阶段后，进入REPLY阶段。

5.5 REPLY

到达此阶段，说明共识已达成，**运行客户端的请求操作o**，并返回<REPLY, v, t, c, i, r>给客户端。其中v是视图编号，t是时间戳，i是副本节点的编号，r是请求执行的结果。

如果客户端收到**f+1不同节点**返回的**相同REPLY消息**，说明客户端发起的请求已经达成共识，否则如果客户端没有在有限时间内收到足够的回复，客户端将判断**是否再次向所有副本节点进行广播请求。**

为什么客户端收到f+1个不同节点返回相同的REPLY消息，就能认为达成共识了呢？因为我们假设的拜占庭节点(恶意节点或失效节点)有f个，f+1个相同的REPLY消息说明其中至少有一个好节点返回正确的结果了，好节点返回正确的结果，那么就可以认为消息已经得到有效的共识。

如果**客户端没有在有限时间内收到回复**，请求将**向所有副本节点进行广播。**如果请求已经在副本节点处理过了，副本就向客户端重发一遍执行结果**。如果请求没有在副本节点处理过，该副本节点将把请求转发给主节点。如果主节点没有将该请求进行广播，那么就有认为主节点失效，如果有足够多的副本节点认为主节点失效，则会触发一次视图变更。**

> 另外：根据副本发给客户端的响应为<REPLY,v,t,c,i,r>，客户端通过p = v % |R|可以推导出主节点的编号。以后客户端就能够通过点对点消息向它自己认为的主节点发送请求，然后主节点自动将该请求向所有副本节点进行广播。

5.6 视图切换(View Change)

1. **触发条件**

视图改变由以下两个条件之一触发：

- 副本从一个客户得知，主节点存在不正当行为（例如：伪造数据等）
- 副本不能收到主节点发出的消息

> View Change由副本节点发起，它们向其他副本节点发送 IHatePrimary 消息以启动一个视图改变。

2. **发送VIEW - CHANGE消息的条件(需理解)**

副本持续等待接收 IHatePrimary 消息，直到遇到下面两个条件之一停止等待：

- 当接收到超过f+1个节点的 IHatePrimary 消息
- 如果收到了其他节点的 VIEW - CHANGE 消息。

当遇到这两个条件之一时，将会广播一条<VIEW-CHANGE, v+1, n, C, P, i>消息，n是最新的stable CheckPoint的编号(n之前的消息已经被执行完了，需要在日志中清除)，C是由2f+1个节点验证过的CheckPoint消息集合，P是当前副本节点未完成的请求的PRE-PREPARE和PREPARE消息集合。v+1 是新的视图编号。

3. **NEW-VIEW的条件 (需理解)**

当节点收到 2f 个有效的VIEW-CHANGE消息后，向其他节点广播<NEW-VIEW，v+1, V, O>消息。V是有效的VIEW-CHANGE消息集合。O是主节点重新发起的未经完成的PRE-PREPARE消息集合。PRE-PREPARE消息集合的选取规则：

- 选取V中最小的stable CheckPoint编号min-s，选取V中prepare消息的最大编号max-s。
- 在min-s和max-s之间，如果存在P消息集合，则创建<<PRE-PREPARE, v+1, n, d>, m>消息。否则创建一个空的PRE-PREPARE消息，即：<<PRE-PREPARE, v+1, n, d(null)>, m(null)>, m(null)空消息，d(null)空消息摘要。

副本节点收到主节点的NEW-VIEW消息时(**谁先发送NEW-VIEW消息，谁更可能成为新的主节点**)，验证其有效性，有效的话，进入v+1视图，并且开始O中的PRE-PREPARE消息处理流程。

5.7 垃圾日志清理

在上面的流程中我们看到，消息的各个环节都会记录日志，这将占用大量空间，所以当请求执行请求后，需要把之前记录的该请求的信息清除掉。具体过程如下：

每执行完一条请求，该节点会再一次发出广播，就是否可以清除信息在全网达成一致。**更好的方案是**，执行K条请求后再向全网发起广播，告诉大家它已经将这K条执行完毕；如果大家反馈说这K条我们也执行完毕了，那就可以删除这K条的信息了；接下来再执行K条，完成后再发起一次广播，即每隔K条发起一次全网共识，这个概念叫**checkpoint**，即每隔K条去征求一下大家的意见，要是获得了大多数**(2f+1)**的认同（a quorum certificate with 2 f + 1 CHECKPOINT messages (including its own)），就形成了一个 stable checkpoint（记录在第K条的编号）。

这是理想的情况，实际上当副本节点 i 向全网发出checkpoint共识后，其他节点可能没有执行完这K条请求，所以副本i不会立即得到响应，它还要继续自己的事情，那这个checkpoint在它那里就不是stable的(因为该副本节点又执行了其他消息请求)。这里有一个处理策略：**对该副本来说，它的低水位h等于它上一个stable checkpoint的编号，高水位H=h+L（一般我们设置L是K的倍数，例如2倍）**，这样即使该副本节点处理速度很快，它处理的请求编号达到高水位H后也得停一停自己的脚步，直到它的stable checkpoint发生变化，它才能继续向前读取并执行新的消息请求。

六、算法论证

6.1 主节点宕机，集群是否能正产工作?

集群中各节点间通过**心跳包**监听节点状态，如果leader宕机，那么副本节点将无法收到主节点的心跳，超过一定时间后，副本节点将发起发送IHatePrimary消息给其他节点，以触发View Change。

6.2 如果副本节点宕机，集群是否能够正常工作?

可以，如流程图中的示例。

6.3 如果主节点伪造请求内容，集群数据是否能继续保持可靠?

- 情况一：主节点分别发给副本节点不同的结果，让集群始终无法达成共识。当发生这种情况时，副本节点因为无法达成共识，认为主节点作恶，此时可以发起「View Change」流程。
- 情况二：假设客户端输入内容是1，主节点将内容改成2后分别发给所有的副本节点，因为副本节点收到的数据都是一致的，可以达成共识（决策值是2），出现这种情况时，集群是否能感知并保持数据可靠?

> 集群是可以保持可靠的，原理如下：当客户端发送Request给主节点时带有摘要信息（d），这个摘要在整个共识阶段都会用到，并且被所有节点验证，并且最后REPLY给客户端的时候也会返回此信息。如果整个过程中d一致，那么说明数据是可靠的。如果当客户端收到REPLY中此信息和发送时不一致，则认为数据被篡改了，客户端就会向所有副本发送一条IHatePrimary消息来通知整个不正当的行为，这将导致触发「View Change」。

6.4 View Change后，之前的数据是否会丢失?

在视图改变时，消息中将包含视图最近一次提交的消息，这将用来恢复之前正确历史，见View Change部分。

6.5 所有节点上数据的顺序是否能保持一致?

PBFT使用了三阶段协议（预准备、准备、确认）实现。预准备和准备两个阶段用来确保同一个视图中请求发送的时序性(每条消息请求都有编号n)；准备和确认两个阶段用来确保在不同的视图之间的确认请求是严格排序的。

预准备阶段给请求分配了一个序号n，准备阶段如果验证通过，那么这个消息将持久化下来。这可以确保其他节点中持久化下来的消息顺序与主节点一直，即实现同一个视图内的顺序性。

消息经过准备、确认阶段后，说明消息已经达成共识了，并且已经持久化下来了，消息持久化后，顺序也就确定了；即便此时发生了view change，也可以保证消息可靠。

需要注意的是，这里消息的顺序由主节点接受消息分配序号n为准，并不以客户端请求顺序为准，因为客户端有可能后发出的请求，主节点先收到。