mp.weixin.qq.com

分布式系统一致性发展史（二）|

网络模型和故障模型

> 导语：
>
> 在本系列第一篇文章中我们提到

> 了Lamport Clock如何启发人们在分布式系统中开始使用新的的思维方式, 并介绍了Sequential Consistency和Linearizability. 本

> 篇为分布式系统一致性发展的第二篇长文，会介绍他们所延展出来的一些应用问题，让大家更好的理解这两种一致性模型。

在本篇正式开始之前，我们先定义一下分布式系统中的网络和故障的模型，这部分稍微有点枯燥，但很重要，了解它们才能继续研

> 究更深层的问题。

分布式系统中的网络模型

同步网络 (synchronous network)

这里的同步网络和编程中的同步阻塞io和异步非阻塞io是两回事，不要弄混了。

同步网络是指：

i). 所有节点的时

钟漂移有上限

ii). 网络的传输时间有上限

iii). 所有节点的计算速度一样.

这意味着整个网

络按照round运行, 每个round中任何节点都要执行完本地计算并且可以完成一个任意大小消息的传输. 一个发出的消息如果在一

个round内没有到达, 那么一定是网络中断造成的, 这个消息会丢失, 不会延迟到第二个round到达. 在现实生活中这种网络比

较少，尽管很少，同步网络仍然是在计算机科学中是不可缺少的一个模型，在这种模型下可以解决一些问题，比如拜占庭式故障.

但我们每天打交道的网络大多都是异步网络.

异步网络(asynchornous network)

和同步网络相反,

节点的时钟漂移无上限，消息的传输延迟无上限，节点计算的速度不可预料。这就是和我们每天打交道的网络类型。在异步网络中，

有些故障非常难解决，比如当你发给一个节点一个消息之后几秒钟都没有收到他的应答，有可能这个节点计算非常慢，但是也可

能是节点crash或者网络延迟造成的, 你很难判断到底是发生了什么样的故障.

Fault, Error and Failure

这不是绕口令,

你一定要区分他们的关系. 过去很多时候这些词汇混用导致很多问题, 后来统一了这几个词的定义:

Fault: 在系统中

某一个步骤偏离正确的执行叫做一个fault, 比如内存写入错误, 但是如果内存是ECC的那么这个fault可以立刻被修复, 就不会导

致error.

Error: 如果一个fault没能在结果影响到整个系统状态之前被修复,结果导致系统的状态错误,那么

这就是一个error,比如不带ECC的内存导致一个计算结果错误.

Failure: 如果一个系统的error没能在错误状态传

递给其它节点之前被修复, 换句话说error被扩散出去了, 这就是一个failure.

所以他们的关系是fault导致error,

error导致failure.
在分布式系统中,
每个节点很难确
定其它节点内部
的状态, 通常只
能通过和其他节
点的交互监测到
failure. 接下来我

们所说的故障一般都是指failure.

分布式系统中的故障模型

在分布式系统中，故障可能发生在节点或者通信链

路上，下面我们按照从最广泛最难的到最特定最简单的顺序列出故障类型:

byzantine failures: 这是最

难处理的情况，
一个节点压根就
不按照程序逻辑
执行，对它的调
用会返回给你随
意或者混乱的结
果. 要解决拜占
庭式故障需要有

同步网络, 并且故障节点必须小于1/3, 通常只有某些特定领域才会考虑这种情况通过高冗余来消除故障. 关于拜占庭式故障你现

在只要知道这是最难的情况, 稍后我们会更详细的介绍它.

crash-recovery failures: 它比byzantine类故障

加了一个限制，
那就是节点总是
按照程序逻辑执
行，结果是正确
的. 但是不保证
消息返回的时间.
原因可能是crash
后重启了，网络

中断了，异步网络中的高延迟.
对于crash的情况还要分健忘(amnesia)和非健忘的两种情况.
对于健忘的情况，是指这个crash的

节点重启后没有
完整的保存crash
之前的状态信息,
非健忘是指这个
节点crash之前能
把状态完整的保
存在持久存储上,
启动之后可以再

次按照以前的状态继续执行和通信.

omission failures: 比crash-recovery多了一个限制,

就是一定要非健忘. 有些算法要求必须是非健忘的. 比如最基本版本的Paxos要求节点必须把ballot number记录到持久存储中,

一旦crash, 修复之后必须继续记住之前的ballot number.

crash-stop failures: 也叫做crash failure或者

fail-stop failures,
它比omission
failure多了一个
故障发生后要停
止响应的要求.
比如一个节点出
现故障后立即停
止接受和发送所

有消息，或者网络发生了故障无法进行任何通信，并且这些故障不会恢复. 简单讲，一旦发生故障，这个节点 就不会再和其它节点有

任何交互. 就像他的名字描述的那样, crash and stop.

分布式系统中的故障类型还有其他的分类方法,

有些分类会把omission去掉, 有些会加入performance failures, 有些会把crash-stop和fail-stop根据故障检测能力区分开,

此处介绍的是使用较为广泛的一种分类方法. 它们的关系如下:

这四种故障中,拜占庭式故障是

非常难以解决的，Leslie Lamport证明在同步网络下，有办法验证消息真伪，故障节点不超过1/3的情况下才有可能解决. 在现实中，

这类问题解决成本非常高，只有在非常关键的领域会考虑使用BFT(Byzantine Fault Tolerance)的设计. 比如NASA的航天飞

机有5台可以抵抗各种射线影响的AP-101系列计算机，其中四台使用同样的软件运行，另外一台独立运行另外一个独立编写版本

的软件. 空客A320有7台计算机, 分别是三种硬件上运行的三套独立编写的软件. 美国海军的海狼级核动力攻击型潜水艇

(SSN-21)也采用了多组计算机控制. 绝大多数应用是不太考虑重力加速度和射线辐射对硬件的影响的, 稍后本文会介绍拜占庭将

军问题来具体解释一下这类问题. 大多数分布式应用主要是关注crash-recovery的情况, 而crash-stop是一种过于理想化的情况,

后面我们在介绍Paxos算法的时候会给大家讲解为什么这个过于理想化的故障模型会带来什么样的问题.

Consensus问题

之所以要介绍Consensus问题是因为Consensus问题是分布式系统中最基础最重要的问题之一，也是应用最为广泛的

问题，他比其他的分布式系统的经典问题比如self-stabilization的实际应用要多，我们可以通过介绍Consensus问题来更加深入得

介绍一下之前提到的Linearizability和Sequential Consistency.

Consensus所解决的最重要的典

型应用是容错处理(fault tolerannce). 比如在原子广播(Atomic Broadcast)和状态机复制(State Machine

Replication)的时候, 我们都要在某一个步骤中让一个系统中所有的节点对一个值达成一致, 这些都可以归纳为Consensus问题.

但是如果系统中存在故障, 我们要忽略掉这些故障节点的噪音让整个系统继续正确运行, 这就是fault tolerance. Consensus问题

的难点就在于在异步网络中如何处理容错.

Consensus问题的定义包含了三个方面, 一般的Consensus问题

定义为:

- **termination:** 所有进程最终会在有限步数中结束并选取一个值, 算法不会无尽执行下去.

- **agreement:** 所有非故障进程必须同意同一个值.

- **validity:** 最终达成一致的值必须

是V1到Vn其中一个，如果所有初始值都是vx，那么最终结果也必须是vx.

Consensus要满足以下三个方面:

termination, agreement 和 validity. 这三个要素定义了所有Consensus问题的本质. 其中termination是liveness的保证,

agreement和validity是safety的保证, 分布式系统的算法

liveness和safety就像一对死对头, 关于liveness和

safety的关系, 我们将会在本系列后面的文章中介绍. 所有需要满足这三要素的问题都可以看做是Consensus问题的变体.

在异步网络中，
如果是拜占庭式
故障，那么Paxos
和Raft也无法解
决这一类问题，
严格讲这是没有
办法解决的，很
长一段时间内我

们只看到在航天和军事领域通过同步网络解决此类问题. (直到Babara Liskov在2002年提出PBFT我们才可以在放松

liveness的情况下解决此类问题，为此Barbara Liskov获得了图灵奖. 我们可能会在将来的文章中介绍PBFT). 对于一般的应用来

说拜占庭故障出现的概率太低而解决的成本实在是太高, 所以我们一般不考虑拜占庭式故障. 我们主要是关注crash-recovery

failure的模型下的异步网络. 这种情况下根据FLP理论, 只要有一个故障节点, Paxos/Raft都是有可能进入无限循环而无法结束

的, 但是实际上
这个概率非常低,
如果放松
liveness的要求,
我们认为这种情
况下Paxos/Raft
是可以解决的.
以下介绍

Consensus问题的时候我们都不考虑拜占庭式故障, 我们的故障模型是crash-recovery failures, 网络模型是异步网络.

在同步网络中因为所有节点时间偏移有上限，所有包的传输延迟也有上限，节点会在一个round内完成计算并且传输完成，所以

一旦超过一定时间还没有收到返回的消息，我们就可以确定要么网络中断要么节点已经crash. 但是我们现实当中都是异步网络，

传输延迟是没有固定上限的，当很长时间一个节点都没有返回消息的时候，我们不知道是这个节点计算速度太慢，还是已经crash

了. 如果是这个节点计算太慢, 超时之后, 过了一会这个节点又把结果再发回来了, 这就超出crash-stop故障模型的范围了,

这种情况需要用crash recovery的模型来解决.

在异步网络中无法区分crash和包延迟会导致consensus问题非常难解决。

今天是分布式系统发展史第二篇的第一部分内容，明天我们将继续发布第二部分内容《两阶段提交和三阶段提交的发展史》，

敬请期待。

如果您想投稿给我们，或者想转发和采用我们的稿件，请回复“合作”，小编会在2小时内回复您的

投稿和合作需求。

本文作者：Daniel，吴强， 现任点融网首席社交平台架

构师，前盛大架构师，专注分布式系统和移动应用，有十三年的开发经验，目前在点融做最爱的两件事情：写代码和重构代码。

随着新一轮融资，点融网开始了大规模的扩张，需要各种优秀人才的加入，如果您觉得自己够优秀，欢迎加入我们！