高级操作系统

3. 分布式进程与处理机

3.1 系统模型

熊焰，yxiong@ustc.edu.cn
黄文超, huangwc@ustc.edu.cn
http://staff.ustc.edu.cn/~huangwc/advancedos

参考书目：《分布式操作系统》，《分布式系统原理与范型》

本章内容

1. 分布式系统模型
2. 分布式处理机分配
3. **分布式进程调度**
4. **分布式系统容错**
5. 实时分布式系统

3. 分布式进程调度

- 常见方式：每个处理机只是进行自己的本地进程调度，而不管其它处理机正在干什么
- 问题：某些情况下很低效
  - 比如： 一组相关的、彼此需要通讯的进程在不同的处理机上运行

在分布式操作系统中，每个处理机只是进行自己的本地进程调度（假定它上面有多个进程在运行），而不管其它处理机正在干什么。在大多数情况下，这种方式工作得很好。但是，当一组相关的、彼此需要通讯的进程在不同的处理机上运行，那么，各自独立的进程调度就不是最有效的方法了。

3. 分布式进程调度

- 例：
  - A与B运行在一个处理机上
  - C与D在另一个处理机上
  - A要给D发送多个消息
    - 一个消息交换需要花费2个时间片

处理器机

时间片	0	1
0	A	C
1	B	D
2	A	C
3	B	D
4	A	C
5	B	D

我们可以用一个例子来说明独立本地进程调度所产生的问题。进程A和进程B运行在一个处理机上，而进程C和进程D在另一个处理机上运行。每一个处理机都是以100毫秒为一个时间片进行分时处理的。进程A和C在奇数时间片中运行，而B和D在偶数时间片中运行，假定进程A要给D发送多个消息。在时间片1，A启动运行并且立即发送一个消息给进程D，但C正在运行，而D不在运行。在100毫秒之后进程切换了，进程D被调度运行，它接到了进程A发来的消息并立即回送一个应答消息。因为进程B正在运行，故进程A只有在100毫秒之后才能收到 这个应答消息。这就是说，一个消息交换需要花费200毫秒的时间。因此，需要一个协同进程调度算法来保证相互通信的进程能够同步执行。

3. 分布式进程调度

- 解决方法：
  - 同时启动一组相互联系的进程
- John K. Ousterhout: Scheduling Techniques for Concurrebt Systems. ICDCS 1982: 22-30, 引用次数：703

尽管动态地确定进程之间的通信比较困难，但在大多数情况下，我们可以同时启动一组相互联系的进程。例如，通常UNIX管道中过滤器之间的通信比它们与其它进程之间的通信要多。我们假定进程都是成组创建的，并且组内进程之间的通信要比组间进程之间的通信多得多。我们可以进一步地假定系统有足够多的处理机来处理最大的一组进程，并且每一个处理机都是具有N个时间片的多进程处理机。

Ousterhout在1982年提出了一个基于协同调度概念的算法，该算法在调度时考虑了进程间的通信以此来保证同一组中的所有进程都在同一个时间片内同时运行。

3. 分布式进程调度

处理机

时间片	0	1	2	3	4	5	6	7
0	×				×			
1			×			×		
2		×			×		×	
3		x	x	x	x			
4		×		×				×
5			×		×			

该算法使用了一个概念上的矩阵，每一列表示一个处理机上的进程表，如图3-1所示。因此，第四列表示在处理机4上运行的所有进程。第三行表示不同处理机上在时间片3内运行的进程。算法基本思想是每一台处理机都使用循环调度算法。例如，如果处理机0启动在时间片3内运行的进程，那么，所有的处理机也必须启动时间片3内运行的进程（如果有的话）。算法的主要目的是所有的处理机都首先运行在时间片0内运行的进程，然后，同时运行在时间片1内运行的进程，依此一直运行下去。算法可以用一个广播消息来通知所有处理机在何时进行进程切换，以便保证时间片的同步。将同一组内所有进程都放在不同处理机上同一个时间片内并保证同一组内的所有进程同时被调度运行这样可以获得N倍的并行度，并使通信吞吐率达到最大。

在图3-1中，为了提高性能，4个必须通信的进程应该分别放入处理机1、2、3、和4的时间片3中。这种调度方法可以与MICROS使用的层次式进程管理算法结合起来使用，只要每一个系主任都保存他属下教师的矩阵，对矩阵中的进程分配时间片，并广播时间片同步消息即可。

4. 分布式系统容错

- 当一个系统没有完成它应该完成的任务，我们将其称之为**失败**或者**失效**
- 动机 (场景):
  - 超市的分布式订购系统
  - 分布式航空交通指挥系统
  - 安全要求较高的场合等

当一个系统没有完成它应该完成的任务，我们将其称之为失败或者失效。在某些情况下，例如，在超市的分布式订购系统中，一个失败有可能导致罐装豆子缺货。而在另一些情况下，例如在一个分布式航空交通指挥系统中，一个失败可能会导致一场巨大的灾难！随着分布式计算机系统广泛应用于安全要求较高的场合，防止失败就变得越来越重要了。

4. 分布式系统容错
4.1 部件错误

- 错误的种类
  - 偶发性错误
  - 间歇性错误（危害较大）
  - 永久性错误

错误一般被分为三类：
偶发性错误：偶尔发生一次，然后再也不会发生的错误。再重复操作一次，错误就会消失。例如，一只鸟飞过微波传输器所发射的光束可能导致网络中一些数据位的丢失，但传输超时后再传输时，网络数据的传输又恢复正常。
间歇性错误：一会儿发生一会儿消失，反复不断。例如，网线接触不良会造成网络一会儿连通一会儿断连，这种间歇性错误很难发现，因而危害性比较大。特别是当人们使用各种手段进行诊断时，故障又消失了。
永久性错误：当错误出现时，它是不会自动消失的，必须将发生错误的部件修复后，错误才能消失。例如，芯片烧坏、软件错误以及磁盘磁头损坏等都属于永久性错误。

4. 分布式系统容错
4.1 部件错误

- 设计和制造容错系统的目的
  - 保证即使存在一些错误，整个系统仍然能够正常地工作。
- 这个目标与设计各个独立可靠的部件是**不相同**的，因为它允许当**一些部件失效时**，系统仍能正常运行。
- 错误或失效可能在任何一个层次发生
  - 晶体管、芯片、电路板、处理机、操作系统、用户程序等等

设计和制造容错系统的目的：保证即使存在一些错误，整个系统仍然能够正常地工作。这个目标与设计各个独立可靠的部件是不相同的，因 为它允许当一些部件失效时，系统仍能正常运行。
错误或失效可能在任何一个层次发生：例如，晶体管、芯片、电路板、处理机、操作系统、用户程序等等。

4. 分布式系统容错
4.1 部件错误

- 如果一个部件在一秒钟时间内发生错误的概率为p，那么，它连续k秒正常工作后发生错误的概率是$p(1-p)^k$，失败发生的均值由下面的公式给出：

$$发生错误的平均时间 = \sum_{k=1}^{\infty} kp(1-p)^{k-1}$$

- 发生错误的平均时间=1/p

- 例如，如果一个部件发生错误的概率是每秒10^{-6}，那么，发生错误的平均时间就是10^6，也就是11.6天！

在容错系统中，需要对部件发生的错误进行分析和统计。如果一个部件在一秒钟时间内发生错误的概率为p，那么，它连续k秒正常工作后发生错误的概率是p(1-p)k，失败发生的均值由下面的公式给出：

(见PPT)

从k=1开始，使用级数无限求和公式：Σα^k=α/(1-α)，其中，α=(1-p)，两边再对p求导数，并乘以-p，我们可以得到：

发生错误的平均时间 = 1/p

例如，如果一个部件发生错误的概率是每秒10^-6，那么，发生错误的平均时间就是10^6，也就是11.6天！

4. 分布式系统容错
4.2 系统错误

- 部件错误—>系统错误
  - 人们希望**系统**能够在一些**部件发生错误**时仍能继续正常工作，而**不仅仅一味地避免**发生部件错误

通常，在一个分布式系统中，人们希望系统能够在一些部件发生错误时仍能继续正常工作，而不仅仅一味地避免发生部件错误。由于分布式系统中有很多的部件，所以，部件发生错误的可能性比较大，因此，对于一个分布式系统来说，系统的可靠性是非常重要的。

我们将要讨论处理机发生错误或崩溃的情况，由软件引起的进程错误和崩溃情况也基本上相同。

4. 分布式系统容错
4.2 系统错误

- 系统错误种类
  - Fail-silent 错误
  - Byzantine错误

Fail-silent错误：出错的处理机仅仅是停止运行，并对接下来的输入既不响应也不产生输出，从而表示它停止工作了，它也称为fail-stop错误。

Byzantine错误 :出错的处理机仍然继续工作，但对输入产生错误的响应，甚至与其它出错的处理机一起产生更严重的错误，它们的特征是看起来好像都在正常工作。显然，处理Byzanti -ne错误比处理Fail-silent错误要难得多。

4. 分布式系统容错
4.3 同步系统与异步系统容错

- 第三种错误分类（抽象层次）
  - 在时间T内没有应答
- 显然，异步系统比同步系统更难对付

第三类错误：假定系统中一个处理机给另一个处理机发送一个消息，那么，必须在给定的时间T内得到一个回答，如果没有得到一个回答，则发送处理机就认为接收处理机已经崩溃。时间T必须包含处理消息丢失重发的时间。

同步系统的性质：系统总能在一个确定的时间内对一个消息做出响应。不具有这个性质的系统就成之为异步的。很不巧，这个术语与已经被使用的术语相冲突。
显然，异步系统比同步系统更难进行容错处理。如果一个处理机发送一个消息，并且知道如果在给定时间T内没有收到回答的话，那么，就意味着消息接收者已经崩溃，然后，发送处理机就可以相应地采取一些正确措施。如果不知道响应时间的上界，那么，如何确定一个系统是否已经崩溃将是一个很麻烦的问题。

4. 分布式系统容错
4.4 方案：冗余容错

- 信息冗余
  - 如：海明码
- 时间冗余
  - 重复执行（解决偶发性错误或间歇性错误）
- 物理冗余
  - 增加额外的物理设备

通常，解决容错的常用方法就是使用冗余，冗余有下列三种：
信息冗余：信息冗余就是给数据添加一些额外的信息位，以便检查数据出错时可以迅速将其纠正过来。例如，海明码可以添加在传输的数据中，并可以校正传输线上由噪音所造成的错误。
时间冗余：在一个事务执行之后，如果需要，可以再次被执行一次。换句话说，如果一个事 务处理失败，则它仍然可以再次重复执行一次，并且无任何副作用。时间冗余特别适用于解决偶发性错误或间歇性错误。
物理冗余：增加额外的物理设备使系统能够允许一些部件的出错或失效。例如，由于系统中有多个冗余的处理机，使得系统能够在一些处理机崩溃之后，仍然能够继续正常运行。

4. 分布式系统容错
4.4 方案：冗余容错

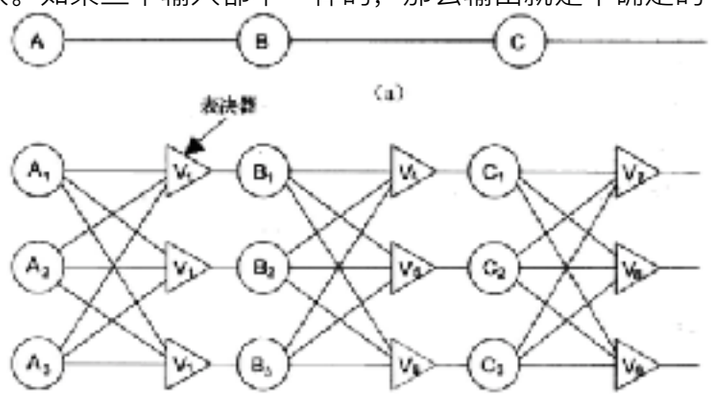
- 两种组织冗余处理机的方法
  - **主动复制** (active replication)
    - 多个处理机完全并行地同时工作，其中一部分处理机失效后，其它处理机继续工作。
  - **主备份** (primary backup)
    - 一个处理机作为服务器，只有当它失效后，才用另一个备份处理机去代替它。

4. 分布式系统容错
4.5 方案：主动复制

- 上述两种冗余处理机的方法都需要考虑以下几个问题:
  - 复制的程度。
  - 不出错时，平均情况和最坏情况下的性能。
  - 出错时，平均情况和最坏情况下的性能。

4. 分布式系统容错
4.5 方案：主动复制

- 主动复制的典型案例
  - 三模冗余(TMR=Triple Modular Redundancy)
    - 每一个设备都复制了三个。在每一级电路中都有三个表决器。每一个表决器都有三个输入和一个输出。如果两个或三个输入相同，那么输出就等于输入。如果三个输入都不一样的，那么输出就是不确定的



主动复制是一种使用物理冗余来提供容错的技术。这种技术已应用于许多领域，例如，在生物学中，哺乳动物有两个眼睛、两个耳朵、两个肺等等；在航空领域中，747飞机有四个发动机，但只使用三个进行飞行；在体育中，有多名裁判，以防有一个裁判误判等。有一些书的作者把主动复制写作状态机方法(state machine approach)。
主动复制方法已广泛应用于电子集成电路的容错。在如图4-12(a)所示的集成电路中。信号依次经过设备A、B和C。如果其中一个出现问题，那么，最后结果就可能是错误的。

而在图4-12(b)中，每一个设备都复制了三个。在每一级电路中都有三个表决器。每一个表决器都有三个输入和一个输出。如果两个或三个输入相同，那么输出就等于输入。如果三个输入都不一样的，那么输出就是不确定的。 这种设计被之称为三模冗余(TMR=Triple Modular Redundancy)。
假设A2出错了，V1、V2和V3都能得到两路正确（即相同的）的输入以及一路错误的输入，每一个表决器都输出正确值给下一级。因此，A2错误已完全被屏蔽了。因此，B1、B2和B3的输入值与没有发生错误时的输入值完全相同。

除了A2出错以外，如果B3和C1也同时出错的话，那么，这些错误也完全可以被屏蔽，最终的三个输出仍然是正确的
每一个表决器也都是一个部件，它随时也都可能出错。例如，如果V1失效了，那么，B1的输入就是错误的。只要其它部件正常工作，B2和B3的输出仍然是正确的，并且V4、V5和V6都将向第三级输出正确的结果。V1出错和B1出错在效果上是一样的。在这两种情况下，B1都产生错误的结果，但最后都被后面的级屏蔽了。

TMR可以递归地使用，例如，为了提高芯片的可靠性，芯片内部都使用TMR技术，但设计师可以使用多个相同的含有TMR技术的芯片进行更高一级的TMR设计。

主动复制中所有服务器可以看作为一个大的有限状态机：它们接收请求并给出应答。读请求并不改变服务器的状态，但写请求会改变服务器的状态。如果每一个客户请求都被发送给所有的服务器，那么，这些服务器接收这个请求并以同样的方式来处理它，处理完毕之后，无错误的服务器都处于相同的状态，并给出相同的结果。客户端中的表决器可以综合这些结果，将错误的结果屏蔽掉。

4. 分布式系统容错

4.5 方案：主动复制

- 问题：需要多少个复制服务器才能满足容错的要求
- 理想条件：最多k个部件同时出错
  - Fail-silent
    - k+1
  - Byzantine
    - 2k+1
- 实际条件：需要概率分析

一个重要的问题就是需要多少个复制服务器才能满足容错的要求呢？如果在最多k个部件同时出错时系统仍然能够正常工作，那么，这个系统就称之为k容错的。对于Fail-silent类型的错误，系统只要拥有k+1个部件就可以满足k容错的要求。例如，系统中有k+1个处理机，若k个处理机因Fail-silent类型错误而停止工作，那么，剩下的那个处理机仍能够给出正确的结果。

如果处理机的错误属于Byzantine类型，那么，出错的处理机一直处于运行状态并给出错误的结果。这时至少需要2k+1个处理机才能获得k容错。在最坏情况下，k个出错处理机碰巧给出同样的结果。然而，剩下的k+1个未出错的处理机仍然给出相同的正确结果，于是，客户端的表决器仍然可以从大多数结果中获得正确的结果。

虽然，在理论上，一个具有k容错的系统可以通过由k+1个相同的正确结果来屏蔽k个错误的结果。但是，在实际上，人们很难肯定在任何情况下k+1个处理机的相同结果是正确的，而k个处理机的相同结果是错误的。因此，在容错系统中需要使用概率分析。

4. 分布式系统容错
4.5 方案：主动复制

- 讨论：该方案的一个前提
  - **原子广播问题**（atomic broadcast problem）
    - 所有请求到达服务器的**顺序**都相同—>
      - 保证服务器对所有的请求都能以相同的顺序执行
  - 方法1: 所有的请求进行全局顺序编号
    - 问题：需一个服务器来编号
  - 方法2: lamport时钟
    - 问题：难以判断哪个消息是第一个请求

这种有限状态机模型的一个前提就是所有请求到达服务器的顺序都相同，我们称其为原子广播问题（atomic broadcast problem），事实上，这个条件可以稍微放宽一些，因为读操作不会产生原子广播问题而写操作则会。为了解决原子广播问题即保证服务器对所有的请求都能以相同的顺序执行，人们提出了许多方法。一种解决方法就是给所有的请求进行全局顺序编号。所有请求都首先发送到一个全局顺序编号服务器上，由该服务器分配一个全局顺序编号，但必须考虑这个服务器，如果它坏了，我们可以采用内部容错的方法来加以解决。

另一种方法就是使用Lamport逻辑时钟。如果每一个发送到服务器上的请求消息都含有一个邮戳，而所有的服务器都根据请求消息中的邮戳顺序来处理消息。这样，所有的服务器都可以以相同的顺序来处理请求消息。但问题是：当一个服务器收到一个请求消息的时候，它并不知道是否有邮戳更小的请求消息还未到达。实际上，大多数使用邮戳的方法都会碰到这个问题。总之，采用主动复制的方法来解决容错并不是一个简单的事情。

另一种方法就是使用Lamport逻辑时钟。如果每一个发送到服务器上的请求消息都含有一个邮戳，而所有的服务器都根据请求消息中的邮戳顺序来处理消息。这样，所有的服务器都可以以相同的顺序来处理请求消息。但问题是：当一个服务器收到一个请求消息的时候，它并不知道是否有邮戳更小的请求消息还未到达。实际上，大多数使用邮戳的方法都会碰到这个问题。总之，采用主动复制的方法来解决容错并不是一个简单的事情。

4. 分布式系统容错
4.6 方案：主备份

- 基本思想：
  - 在任何时候，一个服务器作为主服务器，它承担所有的工作
  - 如果这个主服务器崩溃了，那么就会有一个备份服务器来取而代之
  - 例：政府中的副总统、航空中的副驾驶员、汽车上的备用轮胎、医院手术室中的备用柴油发电机。。

主备份的基本思想：在任何时候，一个服务器作为主服务器，它承担所有的工作。如果这个主服务器崩溃了，那么就会有一个备份服务器来取而代之。在理想情况下，切换过程必须是透明的即只有客户端的操作系统知道，而用户程序感觉不到。和主动复制方法一样，主备份容错也被广泛应用到各个领域。例如，政府中的副总统、航空中的副驾驶员、汽车上的备用轮胎以及医院手术室中的备用柴油发电机等等。

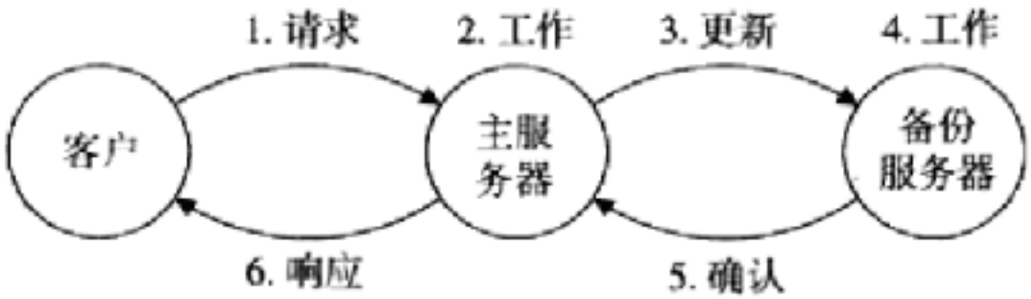
4. 分布式系统容错
4.6 方案：主备份

- 主备份 V.S. 主动复制
  - 不存在原子广播问题
  - 需要的机器非常少, 但是。。。
  - 难以处理byzantine错误
  - 恢复一个崩溃主服务器是非常复杂和耗时的

主备份容错方法与主动复制容错方法相比，有以下两个主要优点和二个缺点：
在通常情况下，请求消息只发往一台主/备份服务器而不是一组服务器，所以，比较简单易行，并且不存在消息顺序问题。
在实际应用中，主备份容错方法需要的机器非常少，因为在任何时候它只需要一台主服务器和一台备份服务器。
当出现byzantine错误时，出错的服务器却认为自己的工作正常有效，造成了一种假象。
恢复一个崩溃主服务器是非常复杂和耗时的。

4. 分布式系统容错
4.6 方案：主备份

- 一个简单写操作协议



- 情况1: 第2步出错

- 情况2: 第4～6步出错

图4-13是主备份容错的一个例子，它描述了一个简单写操作协议。客户端给主服务器发送一个请求消息，主服务器在处理完该请求消息后给备份服务器发送一个更新消息。当备份服务器收到更新消息后，它就进行处理，然后，给主服务器发送一个确认消息。当确认消息到达主服务器后，主服务器才给客户端发送应答。
我们可以发现主服务器在一个远程过程调用(RPC)的不同阶段崩溃时所产生的影响。如果主服务器在处理请求消息之前（即第2步）崩溃，那么，不会产生什么不良的影响。客户端只是在超时之后，再次重新发送请求消息， 直到发送一定次数后，或者因得不到响应而停止发送请求消息，或者它的请求分别得到主服务器和备份服务器的处理，并且只执行一次。如果主服务器在处理完请求消息之后并将更新消息发送给备份服务器之前崩溃了，那么，在备份服务器取代主服务器之后，这个请求消息就会再次从客户端传到主服务器，于是，该请求消息就会被备份服务器处理两次。如果这个处理有副作用的话，那么，就可能产生问题。实际上主服务器在第4步之后第6步之前崩溃的话，那么，这个请求消息都要被执行三次：第2步主服务器执行一次；第4步备份服务器执行一次；在备份服务器取代主服务器之后又执行一次。如果每一个请求消息都带有标志信息，那么，请求消息只被执行两次。一般来说，在主服务器崩溃后，只正确执行一次请求消息的处理是非常困难的。

4. 分布式系统容错
4.6 方案：主备份

- 其他问题：
  - 什么时候在主服务器和备份服务器之间进行切换?
    - 周期性地给主服务器发送测试消息
      - 子问题：区分超时和崩溃？（研究挑战）
  - 接上面问题，如何保证切换后，主服务器必须停止工作?
    - 协商， 但崩溃的主服务器无法协商
    - 使用硬件技术强制停止
    - 双向端口磁盘，但还需要处理磁盘错误

采用主备份容错方法所遇到的一个理论也是一个实际问题就是：什么时候在主服务器和备份服务器之间进行切换。在上面的协议中，备份服务器可以周期性地给主服务器发送测试消息：“你还存在吗？”。如果主服务器在一定时间内没有响应，那么，备份服务器就取而代之。
但是，如果主服务器并没有崩溃，而只是慢一点（例如，在一个异步系统中），怎么办？目前还没有一个有效的方法来区分一个慢主服务器和一个崩溃主服务器之间的差别。
必须保证的是在备份服务器取代主服务器后，主服务器必须停止工作。在理想情况下，在主服务器和备份服务器之间应有一个协议来处理这个问题。然而，与一个已经崩溃的主服务器进行协商是不可能的。最好的解决方案就是采用一种硬件技术强制备份服务器停止取代主服务器工作或重新启动主服务器。此外，我们必须注意到的是：所有主备份容错方法所采用的协议都必须是一致的，这一点是很难办到的。而主动复制容错就不一定需要一致的协议

一个与图4-13略有不同的方法是使用由主服务器和备份服务器共享的双向端口磁盘。当主服务器收到一个请求消息后，它首先把这个请求写入磁盘，然后，它也把结果写到这个磁盘上。这样，主服务器就不需要与备份服务器进行通信了。如果主服务器崩溃了，那么，备份服务器就可以简单地从磁盘中读取信息以得知主服务器崩溃的情况。这个方法的缺点是只有一个共享磁盘。所以，如果这个唯一的磁盘坏了，那么，所有信息都将丢失！当然，磁盘

4. 分布式系统容错
4.7 协作一致性

- 所有的进程都需要协作对某件事达成一致
  - 选择一个协调器
  - 执行一个处理的决定
  - 将任务分配给工作站
  - 同步

在许多分布式系统中，所有的进程都需要协作对某件事达成一致。例如，选择一个协调器、执行一个处理的决定、将任务分配给工作站、同步等等。当通信和处理机都正常工作时，达成协作一致性是很简单的。但当它们工作并不正常时，就会出现许多问题。

分布式协作一致性算法就是让所有未出错的处理机能够对某些问题达成一致性意见，并在有限步数内完成协作操作。

4. 分布式系统容错
4.7 协作一致性

- 它涉及到下列一些问题：
  - 消息传递总是可靠的吗？ （两军问题）
  - 进程会崩溃吗？如果会，它们是Fail-silent错误还是Byzantine错误?
  - 系统是同步的还是异步的?

消息传递总是可靠的吗?
进程会崩溃吗？如果会，它们是Fail-silent错误还是Byzantine错误?
系统是同步的还是异步的?

4. 分布式系统容错
4.7 协作一致性

- “两军问题”(two-army problem)
  - 对于无故障的处理机来说，在通信不可靠的情况下，两个进程要达到协作一致是完全不可能的。

首先看看一个处理机正常而通信线路可能丢失信息的例子即一个著名的“两军问题”(two-army problem)，它说明两个正常处理机在一位信息上达成协作一致性是非常困难的！假设，红军有5000人，驻扎在一个峡谷里。蓝军分为两支，各有3000人，驻扎在可以俯瞰峡谷的周围山上。如果两支蓝军协同进攻红军，那么，他们一定会取得胜利。然而，如果任何单独的一支蓝军去进攻红军，那么，他一定会被消灭。蓝军所要做的就是达成一致同时进攻红军。问题是，他们只能通过一个并不太可靠的途径来通信即派通信兵互通消息，但通信兵常常被红军俘虏。

假定蓝一军的指挥官Alexander将军派通信员给蓝二军的指挥官Bonaparte将军送信“明天清晨6点一起进攻红军，好吗？”。通信兵把信送给蓝二军，Bonaparte将军看完信后回了一封信“好主意，明天早晨6点见。”。通信兵又安全地将回信带回了蓝一军。于是，Alexander将军的军队就开始做进攻的准备。

然而，在这之后，Alexander将军意识到Bonaparte将军并未知道这个通信兵是否安全回到基地，他就有可能不敢发兵。因此，Alex -ander将军就派通信兵去告诉Bonaparte将军，回信已收到，正在做明天进攻的准备工作。

同样，通信兵安全送达蓝一军的确认消息，但，Bonaparte将军又担心Alexander将军不知道他的确认消息是否安全送到。Bonaparte将军会这样推测：Alexander将军可能认为通信兵被俘虏，Bonaparte没有得到他的确认信，也许不敢冒险发兵。于是，Bonaparte将军又让通信兵返回。

尽管每一次通信兵都能安全地送达消息，但是不管他们派多少次通信兵送信，Alexander将军和Bonaparte将军都永远无法达成协作一致。如果某一个协议能在有限步内终止并达成一致，那么，为了最小化这个协议，就必须取消一些多余的步骤。假若某一个消息是使这个最小化协议达成一致的最后一个消息，那么它对协议的一致性来说是至关重要的。如果这条消息无法保证正确到达，那么，整个协议就会失败。

由于两个将军都无法保证最后一条消息能否安全到达即协议不能达到协调一致，所以，其中一个将军就不会冒然出兵。因此，对于无故障的处理机来说，在通信不可靠的情况下，两个进程要达到协作一致是完全不可能的。

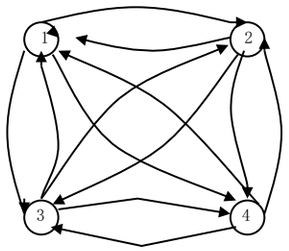
4. 分布式系统容错
4.7 协作一致性

- Byzantine将军问题(Byzantine generals problem)
  - （通信可靠而处理机不可靠）
  - 有m个将军是叛徒
- Leslie **Lamport**, Robert E. Shostak, Marshall C. Pease: The Byzantine Generals Problem. ACM Trans. Program. Lang. Syst. 4(3): 382-401 (1982), 被引用次数：**4592**

现在考虑通信是可靠的而处理机是不可靠的情况。这里，同样有一个经典的与军事有关的例子，叫Byzantine将军问题(Byzantine generals problem)。在这个问题中，红军仍然是驻扎在山谷中，而n支蓝军驻扎在周围的山上。通信是通过点对点的电话来进行的或者是通过收发密码电报来进行的，因此是可靠的。但其中有m个将军是叛徒（即出现Byzantine错误的处理机），他们总是企图通过提供假信息来阻止忠诚将军们达成协作一致。那么，那些忠诚将军之间是否仍然能够达成协作一致呢?

4. 分布式系统容错

4.7 协作一致性



1 Got(1, 2, x, 4)	1Got	2Got	4Got
2 Got(1, 2, y, 4)	(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
3 Got(1, 2, 3, 4)	(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
4 Got(1, 2, z, 4)	(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

1. 每一个将军广播，声明自己军队的人数
2. 将收集的军队人数的结果
3. 每一个将军把自己的向量传给其他每一个将军
4. 每一个将军都要检查所有新收到向量的第i个元素

Lamport等人.在1982年提出了一个可以在特定条件下解决这一问题的递归算法。考虑n=4和m=1的情况，如图4-14所示，算法分四步执行。第一步，每一个将军发送一个可靠的消息给所有其他的将军，声明自己的军队人数。忠诚将军说出的是真实数字，而叛徒告诉每一个将军的数字都各不相同。在图4-14(a)中，我们看到将军1报告他有1K军队，将军2报告他有2K军队，将军3是个叛徒，他分别给出三个不同的人数x、y和z，将军4报告他有4K军队。第二步，将收集的军队人数的结果组成如图4-14(b)所示的向量。

第三步，每一个将军把自己如图4-14(b)所示的向量传给其他每一个将军。在这里，将军3仍然撒谎，编造出12个不同的新值。第三步的结果如图4-14（c）所示。第四步，每一个将军都要检查所有新收到向量的第i个元素。如果有某个数值占多数，那么，这个值就被存入结果向量。如果没有一个数值占多数，那么，结果向量的相应元素就被标为未知(UNKNOWN)。从图4-14(c)中我们可以看到将军1、2和4都能得到协作一致的结果向量(1,2,UNKNOWN,4)。

因而，这个结果是正确的。叛徒的目的无法达到。

Lamport等人在1982年的论文中证明，在一个有m个坏处理机的系统中，仅当系统中还有2m+1个好处理机在正常工作（即系统中共有3m+1个处理机），系统才能达到协作一致。换句话说，要使系统能够达到协作一致，那么至少要有超过三分之二的处理机处于正常工作的状态。

4. 分布式系统容错
4.7 协作一致性

- 理论结果：

  1. (lamport 1982) m个坏处理机~ 2m+1个好处理机

  2. (Fischer 1985) 若系统**异步**且**无传输延迟限制**，则只要有一个处理机崩溃，则系统**不可能达到协作一致**

Lamport等人在1982年的论文中证明，在一个有m个坏处理机的系统中，仅当系统中还有2m+1个好处理机在正常工作（即系统中共有3m+1个处理机），系统才能达到协作一致。换句话说，要使系统能够达到协作一致，那么至少要有超过三分之二的处理机处于正常工作的状态。

然而，糟糕是，Fischer等人在1985年证明：对于一个异步的且无传输延迟限制的分布式系统，只要有一个处理机崩溃（即使是Fail-silent错误），那么，系统都不可能达到协作一致。原因是异步系统对运行慢的处理机和崩溃的处理机不加以区别。