分布式系统期末考点 答案整理

Author： 中山大学 17数据科学与计算机学院 YSY

https://github.com/ysyisyourbrother

1、分布式系统的定义和特点

分布式系统是若干独立自主计算机的集合，这些计算机对于用户来说像单个耦合的系统。分布式具有自主性和耦合性。它的系统特性是：

1. 构成组件并被所有用户共享
2. 系统资源可能不允许访问
3. 软件运行在不同处理器上多个并发进程中
4. 允许多点控制
5. 允许多点失效

2、分布式系统的透明性主要包括哪几方面?

透明性	说明
访问	隐藏数据表示形式的不同以及资源访问方式的不同
位置	隐藏资源所在位置
迁移	异常资源是否移动到另一个位置
重定位	隐藏资源是否在使用过程中移动到另一个位置
复制	隐藏是否对资源进行复制
并发	隐藏资源是否由相互竞争的用户共享
故障	隐藏资源的故障和恢复
持久化	隐藏数据在主存和磁盘这一事实

3、 策略和机制的不同

策略是具体的实现 机制是抽象的设计

4、分布式系统的可拓展性包括哪些方面？有哪些技术可以实现可拓展性

- 规模可拓展性：用户数量和进程数量增加
- 地理可拓展性：节点之间最大物理位置

- 管理可拓展性：管理域的数量

5、 集群系统和网格之间的区别

集群计算系统本质上是通过LAN连接起来的高端计算系统

- 同构： 相同的OS， 近乎相似的硬件
- 单个管理节点

网格计算系统是由各地的节点构成的系统

- 异构
- 包含多个组织
- 容易拓展到广域网环境中

6、云计算作为一种新的计算模式适用于所有企业?

对于公有云环境下，安全性还存在很大问题，因此对安全性或保密性有比较高要求的企业就不能使用。还有如果所需要的硬件设备云计算服务商不提供也不适合。

7、 主要的软件体系结构包括哪几种

8、 主要的系统体系结构包括哪几种？与软件体系结构的主要区别是什么?

确定软件组件、这些组件的交互以及它们的位置就是软件 体系结构的一个实例，称为系统体系结构

集中式体系结构：整个系统包含一个控制中心，协同系统的运行

非集中式组织结构： 系统没有一个整体的控制中心，各个节点独立自主运行

混合组织结构： 系统中既包含集中式结构也包含了非集中式结构；

9、 分布式的分层结构

1. 用户接口层：

   用户接口层包含系统与用户直接交互的单元；例如：显示 管理

2. 应用处理层

   包含应用的主要函数，但是，不与具体的数据绑定

3. 数据层

   数据层管理应用使用的实际数据

10、 非集中化的体系结构主要类型包括哪些?

主要类型分为：

1. 垂直分布： 系统逻辑分层，不同层次分布于不同的机器上
2. 水平分布： 客户或者服务器在物理上分成逻辑上相等的几个部分，每个部分相对独立，且分布在不同的机器上

3. 点对点系统： 水平分布，构成点对点的系统的进程完全相同（既是客户端又是 服务器、无中心化系统）

11、 Chord结构的生成和查找算法；

参看第五章笔记

12、 非结构化的点对点系统搜索内容的方式?

有两种方法：

泛洪：请求节点向它所有邻居节点发出数据搜索请求，如果已经收到过就会被忽略，否则就会在本节点中查找数据项。

随机游走：请求发送节点u从邻居节点中随机选择一个节点，然后进行本地搜索，如果没有完成，就继续随机选一个邻居继续搜索，直到搜索到达目标或终止情况。

13、 什么是超级对等节点？如何确定超级节点?

超级对等节点是**维护一个索引或充当一个代理程序的节点。**在非结构化P2P进行搜索的时候，客户会通过超级对等节点去搜索数据，索引服务器会提高搜索性能，更快定位数据。

客户尽可能归依一个包含客户感兴趣文件索引的超级对等节点，这样可以更快获取用户想要的数据，如果客户发现了一个更好的超级对等体包含更多它想要的数据的索引，或者物理位置更近的，就可以改变当前关系。

14、 在P2P系统中节点之间连接的方式；

P2P系统分为结构化P2P结构、非结构化P2P结构、混合P2P结构。

- 结构化P2P系统将节点组织在一个特定结构的覆盖网络，如环型的Chord结构，
- 非结构化P2P结构的邻接点是随机的，每个节点维护一张动态随机的邻接表构建随机图，边<u,v>存在的概率为P(<u,v>) 。节点周期性的从部分视图中选择别的节点交换信息。
- 混合P2P系统把客户-服务器体系结构和非集中式体系结构组合在了一起。

15、 分布式系统的自我管理；

出现问题能够自我优化、恢复。在需要完成自适应功能时， 系统架构和软件架构之间的界线逐渐模糊。

16、 分布式系统中为什么利用线程而不是进程?

1. 可以避免不必要的阻塞，单线程的进程在IO的时候可能会被阻塞，在多线程的进程中，一个线程被阻塞后，可以切换不同的线程。
2. 可以更好的发挥并行性，一个具有多线程的进程可以在多CPU上并行执行
3. 可以避免进程上下文切换的昂贵开销，切换线程的成本更低。

17、 虚拟机化主要包含哪些方式，简要描述?

1. 进程虚拟机：是一个独立的指令集，在操作系统只上运行一个解释器对指令解释运行比如JAVA的虚拟机就是这样工作的。
2. 原生虚拟机监控器：虚拟机监控器安装在物理硬件上，将硬件拆分成多个虚拟机，在其上安装不同的操作系统
3. 主机虚拟机监控器，需要安装在主机的操作系统之上，然后再其上安装多课不同操作系统。

18、 在客户端-服务器模型中，服务器的状态主要分为几种，简要解释。

服务器分为无状态服务器和有状态的服务器

无状态的服务器不保存客户的信息，也不会告知客户服务器的状态。它不记录一个文件是否被打开过，也不会去追踪用户的信息，性能相对低下。如web服务器就是这样

有状态的服务器会记录客户端的状态信息。它记录客户端打开的文件，也知道客户端缓存了哪些文件，这样可以允许客户端在本地保存共享数据的副本并允许更改。有状态服务器性能高，但存在可靠性的问题。如文件服务器就是这样的。

19、 简述如何实现代码迁移？代码迁移的模型？如何区分强迁移和弱迁移?

在代码迁移模型中，进程由如下三段组成

1. 代码段：包含构成正在运行的程序的所有指令
2. 资源段：包含指向进程需要的外部资源的指针，如打印机等
3. 执行段：存储进程当前执行状态量，比如寄存器、栈等

弱可移动性： 仅仅移动代码和数据片段（**重启执行**）

1. 相对简单，特别是如果代码是可移植的 （目标机器能够运行代码）
2. 需要区分两种模式：代码推送（Push）和代码拉取（Pull）

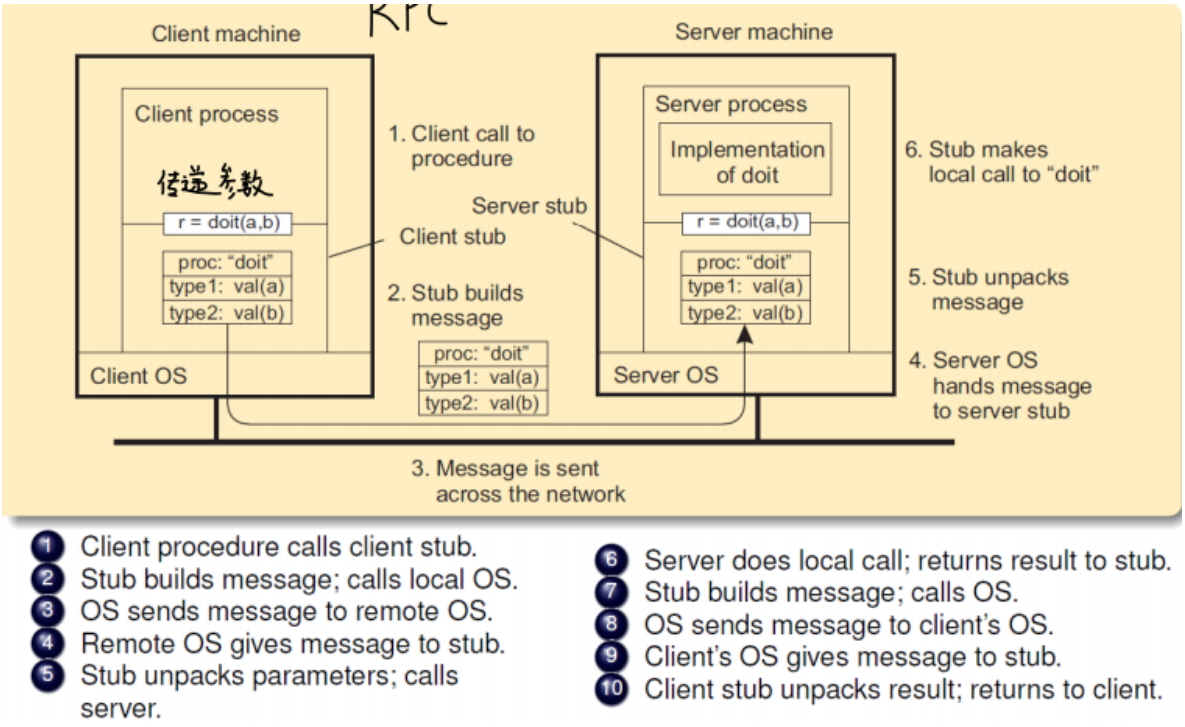
强可移动性： 移动组件，包括执行状态 ，可以从中断位置开始执行

1. 迁移（Migration）：将整个对象从一个机器移动到另外一个机器
2. 克隆（Cloning）：开始克隆，将其设置为相同的执行状态

20、 虚拟机迁移的种类及主要特点?

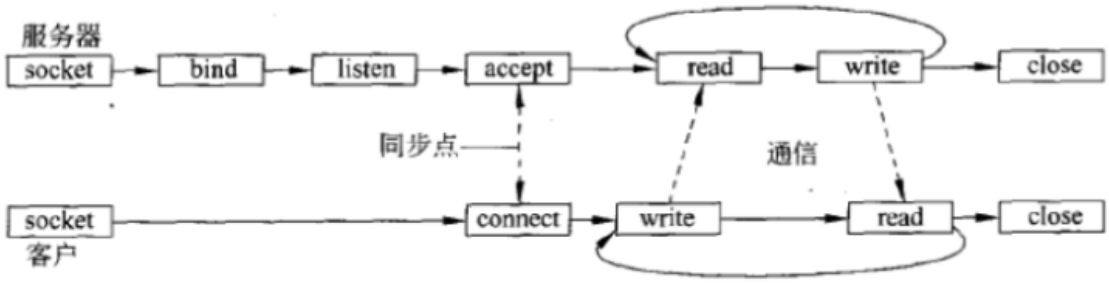
21、 RPC远程过程调用的概念及主要步骤?

远程过程调用就是机器A上的进程调用机器B上的进程，A上的进程被挂起，B上的进程开始执行，调用方通过参数传递的方法将信息传送给被调用方，然后通过传回的结果得到信息。



1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters; calls server.
6. Server does local call; returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client's OS.
9. Client's OS gives message to stub.
10. Client stub unpacks result; returns to client.

22、 使用socket进行网络通信的主要模式及主要过程



23、 什么是点对点通信、广播和多播?

点对点通信实现网络内任意两个用户之间进行信息交换

广播就是将数据发送到指定网络内每一台设备上

多播就是将数据发送到指定网络中的指定多台设备上

24、 简述Gossip数据通信和反熵通信模型?

Gossip 协议利用一种随机的方式将信息传播到整个网络中，并在一定时间内使得系统内的所有节点数据一致。Gossip 其实是一种去中心化思路的分布式协议，解决状态在集群中的传播和状态一致性的保证两个问题。

对传播信息来说，一种技术是依靠感染行为。

Anti-Entropy 类比传染病可以建立消息传播的模型：

具有要传播到其他结点数据的称为：**已感染的**

还没有接受到数据的节点称为：**易受感染的**

不会传播其数据的已更新结点称为：**已隔离的**

反熵模型是 结点P随机选取另一节点Q，然后与Q交换更新信息。

1. P只是把它自己的更新信息发出给Q，即为push的方法
2. P只是从Q那里获得更新信息，即为基于pull的方法
3. P和Q相互发送更新信息给对方，基于push-pull的方法

25、 Chord指纹表的查找过程，节点如何加入和退出指纹表?

见第五章笔记

教程参考： http://www.yeolar.com/note/2010/04/06/p2p-chord/

26、 简述在树结构目录中查询实体及插入实体的过程?

见第五章笔记

27、 名称解析闭包?

知道如何启动和从何处开始启动名称解析通常称为闭包机制

28、 实体的硬链接和软链接?

在命名图中按照特定路径名搜索节点就是硬链接，同一个节点可能存在多条路径到达

软链接又叫符号链接，节点包含了到另一个节点的路径名称。

29、 什么是挂接点和挂载点?

挂接点： 在当前命名空间中用于**存储节点标识符的目录节点** 成为挂接点

挂载点： 外部名称空间中的目录节点称为挂载点；挂载点是 命名空间的“根”

30、 什么是命名空间，命名空间的分层结构主要由哪几部分构成?

- 全局层： 由最高级别的节点构成，即由根节点以及其他逻辑上 靠近根节点的目录节点组成。特点是：稳定，目录表很少改变， 可以代表组织或者组织群
- 行政层： 由那些在单个组织内一起被管理的目录节点组成。行 政层中的目录节点所具有的特点是**代表属于同一组织或行政单 位的实体组**；相对稳定，但是比全局层的目录变化频繁；
- 管理层： 由经常改变的节点组成。如代表本地主机的节点及本 地文件系统等，由终端用户维护；

31、 迭代命名解析和递归命名解析?

见第五章笔记

32、 瞬时同步通信的主要问题以及解决方案?

33、 时钟同步：内同步和外同步；

内同步：精度，保证任意两个机器的时钟之间的偏差在一个特定的范围内。

外同步：准确度，保证当前本地时钟与绝对的物理时钟UTC之间的的差在α内

34、 如何在没有UTC的情况下保障时间的准确性?

Berkeley算法：时间服务器（不一定是UTC时间）周期性地扫描所有服务器，计算时间均值，并告诉所有其他机器如何根据当前时间进行调整。

35、 逻辑时钟？及算法

时间上不一定达成一致，只需要在事件发生顺序上达成一致。

Lamport

36、 什么是全序广播?

在一个分布式系统内，每个个体对自己的副本进行操作后多播这个操作，我们需要保证多播的操作对**每个副本都是按照相同的顺序执行。**

37、 如何利用Lamport逻辑时钟解决互斥访问及临界区访问?

在同一个时刻至多允许一个进程执行的代码片段。多个进程需要在访问顺序上达成一致。（有序访问）

38、 因果有序的多播传播?

使用向量时钟，可以确保所有因果关系先于某个消息m的所有消息接收后才deliver（从消息队列中取出，传送给上层的应用程序）这个消息m。

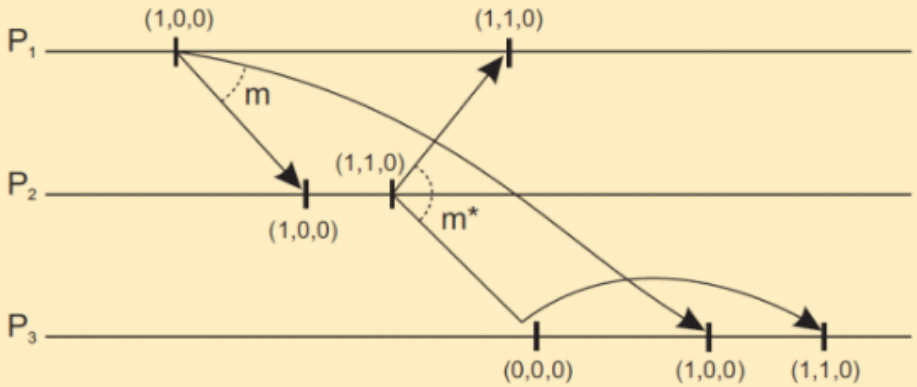
仅当Pi发送消息时才增加VC[i]，当Pj接收到消息后才调整VCj

现在，假设进程 P_j 从进程 P_i 接收一个(向量)时间戳为 ts(m)的消息 m。把该消息传送到应用层将被延时，直到以下两个条件满足为止：

(1) $ts(m)[i] = VC_j[k]+1$

(2) $ts(m)[k] \leqslant VC_j[k]$，其中 $k \neq i$

➢ 强制因果有序的通信



P2向P3发送m时(1,1,0)，P3的向量时钟(0,0,0)，虽满足条件1，但不满足条件2，P3收到m的时候不deliver到应用程序，先缓存到消息队列中或者拒收等待P2重传。等条件满足的时候P3从消息队列中去除消息传递给应用程序。

39、 解决分布式系统中多进程互斥的方法?

- **集中式**

  协作者

- **非集中式**

  N个副本服务器，获得大多数>N/2支持

- **分布式算法**

  Ricart&Agrawala算法：进程要访问共享资源时，构造一个消息，包括资源名、它的进程号和当前逻辑时间。然后发送给所有的其他进程。 进程接收到这条消息，采取响应的动作

  当一个进程接收到来自另一个进程的请求消息时，他根据自己与消息中的资源相关的状态来决定采取的动作

  1. 若接收进程没有访问资源，而且也不想访问资源，向发送者返回 一个OK消息；
  2. 若接收者已获得对资源的访问，那么它就不答应，而是将该请求放入队列中；
  3. 如果接收者想访问资源但尚未访问时，它将收到消息的时间戳与它发送到其他进程的消息的时间戳进行比较。时间戳早的那个进程获胜，如果接收到的消息的时间戳比较早(小），那么返回一个OK消息。如果它自己的消息的时间戳比较早，那么接收者将收到的消息放入队列中，并且不发送任何消息；

  每次需要2(n-1)个消息

  1. 进程0和2要访问，广播消息
  2. 做决策，进程1同时接收到0和2的消息，1不访问资源，分别返回OK给0和2。

     进程0的逻辑时钟8有更高的访问优先权

  3. 进程1访问完了，消息队列里取出进程2的消息，返回OK

  问题：

  - 当一个进程崩溃，不能回答请求，不应答被错误地解释为拒绝请求，阻塞了所有请求进入所有临界区地后续进程。单个故障点被n个故障点所取代， 故障概率提高了 n 倍， 同时网络流量大大增加。
  - 进程维护开销较大，不适用进程数目较多的情况；

- **令牌环算法**

40、 Ricart & Agrawala互斥算法?

分布式算法

Ricart&Agrawala算法：进程要访问共享资源时，构造一个消息，包括资源名、它的进程号和当前逻辑时间。然后发送给所有的其他进程。 进程接收到这条消息，采取响应的动作

当一个进程接收到来自另一个进程的请求消息时，他根据自己与消息中的资源相关的状态来决定采取的动作

1. 若接收进程没有访问资源，而且也不想访问资源，向发送者返回 一个OK消息；
2. 若接收者已获得对资源的访问，那么它就不答应，而是将该请求放入队列中；
3. 如果接收者想访问资源但尚未访问时，它将收到消息的时间戳与它发送到其他进程的消息的时间戳进行比较。时间戳早的那个进程获胜，如果接收到的消息的时间戳比较早(小），那么返回一个OK

消息。如果它自己的消息的时间戳比较早，那么接收者将收到的消息放入队列中，并且不发送任何消息；

问题：

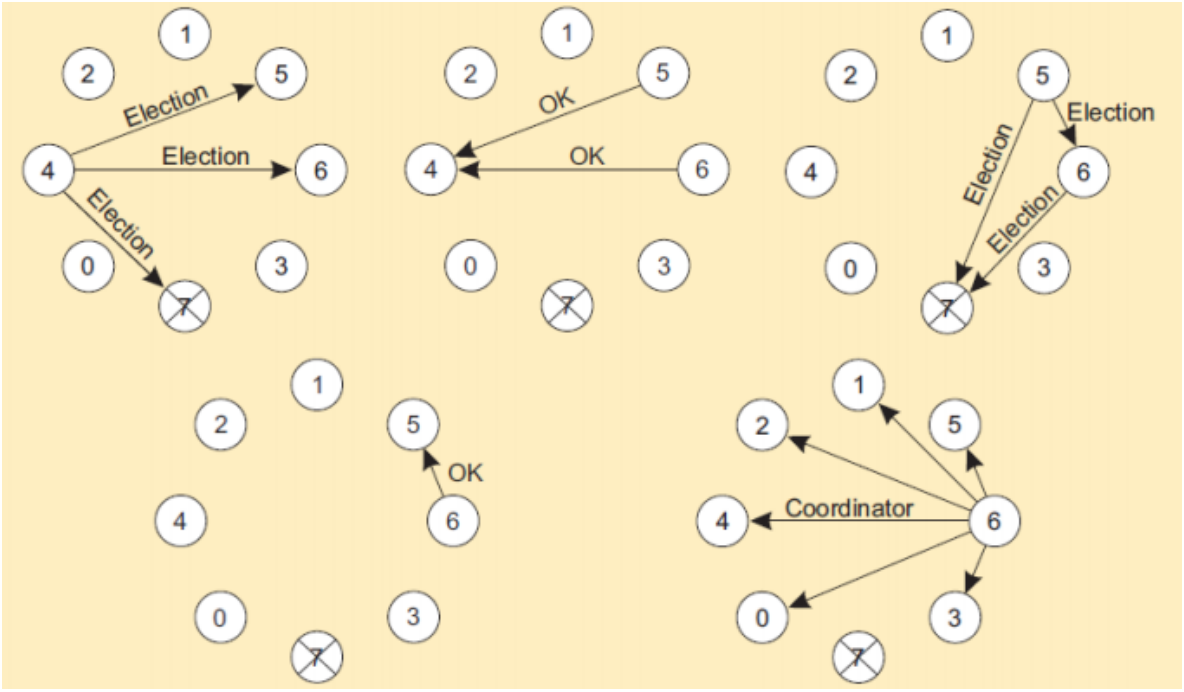
- 当一个进程崩溃，不能回答请求，不应答被错误地解释为拒绝请求，阻塞了所有请求进入所有临界区地后续进程。单个故障点被n个故障点所取代， 故障概率提高了 n 倍， 同时网络流量大大增加。
- 进程维护开销较大，不适用进程数目较多的情况；

41、 无线网络中的选举算法；

bully算法

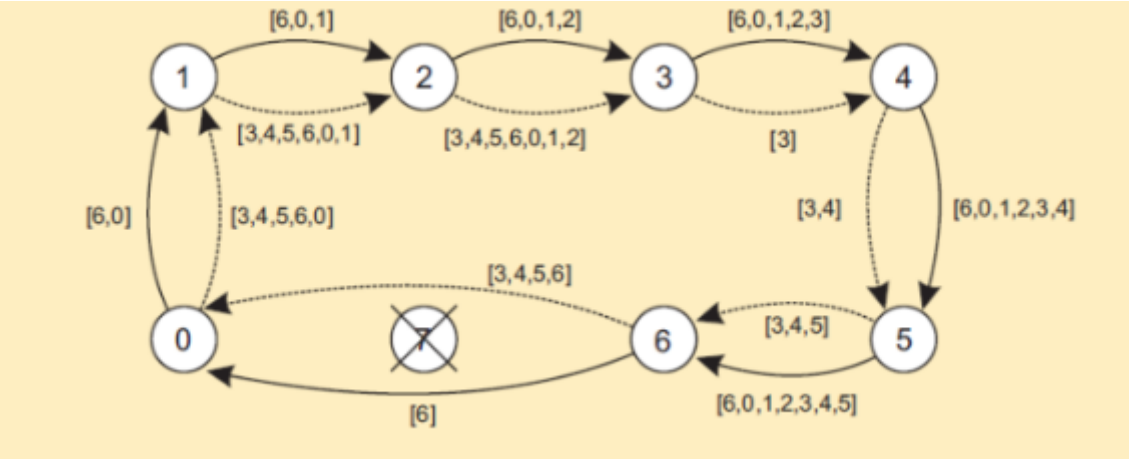
当任何一个进程发现协作者不再响应请求时，他就发起一次选举，进程P按照如下过程主持一次选举：

1. P向所有编号比它大的进程发送一个election信息
2. 如果无人响应，获胜成为协作者
3. 如果有编号比它大的进程响应，则由响应者接管选举工作，P的工作完成。



ring算法

1. 进程按照物理或逻辑顺序进行排序，组织成环形结构。具有最大ID 的进程被选为协作者
2. 当任何进程注意到协作者不工作时，任何进程都可以启动一次选举过程，该进程把选举信息传递给后继 者。如果后继者崩溃，消息再依次传递下去
3. 在每一步传输过程中，**发送者把自己的进程号加到该消息的列表中**， 使自己成为协作者的候选人
4. 消息返回到此次选举的进程，进程发起者接收到一个包含它自己进 程号的消息时，消息类型变成Coordinator消息，并再次绕环向**所 有进程通知谁是协作者以及新环中的成员**都有谁



上面是两个选举过程

42、 数据一致性模型

实质上是**进程和数据存储之间的一个约定**。如果进程同意遵守某些规则，那么数据存储将正常运行。

43、 连续一致性或者一致性程度主要包含哪些方面?

- 数值偏差
- 新旧偏差
- 顺序偏差

44、 什么是顺序一致性和因果一致性?

顺序一致性：任何执行结果都是相同的，就好像所有进程对数据存储的读、写操作是按照某种顺序执行的，并且每个进程的操作按照程序所制定的顺序出现在这个序列中。

因果一致性：所有的进程必须**以相同的顺序看到具有潜在因果关系的写操作**。不同机器上可以以不同顺序看到**并发写操作**

45、 数据为中心的一致性和客户为中心的一致性模型各自适用于什么场景?

以数据为中心的一致性模型： 读写并重的数据存储

以客户为中心的一致性模型： 读多写少，提供弱一致性即最终一致性;

46、 什么是最终一致性? 有什么优缺点?

如果在很长一段时间内没有更新操作，那么所有的副本将逐渐地成为一致的

优点是最终一致性只要求更新操作被传播到了所有副本，实现开销小

缺点是当用户不是每次都连接到同一个副本的时候就会出现问题。可能这一次修改了这个副本，还未达到最终一致性后访问了另外一个副本，用户就会发现不一致性。

47、 读写一致性? 写读一致性? 具体场景?

读写一致性：一个进程对数据项x执行一次写操作的结果总是会被该进程对x执行的后续读操作看见

写读一致性：同一个进程对数据项 x 执行的读操作之后的写操作，保证发生在 与 x 读取值相同或比之更新的值上

48、 服务器副本的放置位置?

分为永久副本、服务器初始化副本和客户初始化副本

49、 内容分发的方式有哪些? Pull-based和Push-based的内容分发方法有什么不同?

内容分发方式:

1. 只传播更新的通知
2. 将数据从一个副本传到另外一个副本
3. 把更新操作传播到其他副本

pull和push

- 基于Push的更新:服务器初始化的方法，不需要其他副本请求更 新，这些更新就被传播到那些副本那里
- 基于Pull的更新:客户端初始化的方法，客户端请求的更新

push劣势：服务器需要记录客户端的数据

pull劣势：更新速度比较慢

50、 在一致性协议中， 如何限定数值偏差? 限制新旧偏差?

限定数值偏差：对每个副本，累加在它身上自己产生的原生操作加和得到v(t)，对每个副本，累加它接收到别的副本传来的操作的和得到vi，要求vt和vi的差距在一个范围内

限制新旧偏差:

- 让服务器 S_k 保持实时**向量时钟** RVC_k，其中RVC_k[i] = T(i)为 到时间 T(i)时， S_k看到了已提交给S_i的所有写操作
- 只要服务器 S_k 通知 T[k]-RVC_k[i] 将超出指定的界限，那么它 就开始拉入来自S_i的时间戳晚于RVC_k[i]的写操作

51、 基于主备协议的复制协议主要包括哪两种方式? 分别有什么特点?

- **远程写协议**

  所有读操作和写操作都转发给单个固定的远程服务器：要在数据项x上执行一个写操作的进程，会把该操作转发给x的主服务器。该主服务器在其x 的本地副本上执行更新操作，随后把该更新转发给备份服务器。

  写操作：经过的比较长，W1-W2-W3-W4-W5

- **本地写协议**

主副本在要执行写操作的进程之间迁移：某个进程对x进行写操作，则定位x的主副本，然后将x转移到自己的位置上。

W3处就已经返回了

在断网之前将数据保存在本地，然后修改是对本地操作的，等重新联网后向原来的主备份同步

52、 如何理解基于团体的复制写协议?

基于团队的协议，读或者写必须经过团体的同意。要保证一致性，需要满足以下的条件。

1. 写团体要占多数
2. 写团体和读团体要有重合

第一个条件是为了防止读写冲突：假设最近的写团体有服务器C-L10个服务器组成，任何随后由三个服务器组成的读团体至少包含一个该集合中的服务器。客户可以选择最新更新过的服务器来读。

第二个条件为了防止写写冲突，否则可能出现两个进程都被允许写

53、 什么是可依赖性？与可依赖性相关的需求包括哪些方面?

含义：软件组件为了给用户提供服务，可能需要来自其他组件的服务（服务依赖），即组件可能依赖于其他组件。

与可依赖性相关的需求

1. 可用性availability：系统已经准备好，马上可以使用。在任何给定时刻，系统都可以正确操作。

   可用就绪的能力 最低99.99 追求 99.9999
2. 可靠性reliability：系统可以无故障地持续运行。根据时间间隔定义。

   高度可靠的系统可以在一个相对长的时间内持续工作而不被中断。

   连续提供服务的能力
3. 安全性safety：系统偶然出故障的情况下能正确操作而不会造成任何灾难。
4. 可维护性maintainability：发生故障的系统被恢复的难易程度

54、 可靠性与可用性的定义及区别?

可靠性为：在0到t时间内系统能正常运行的概率

可用性为：在0到t时间段 组件c可用的时间比例

55、 分布式系统中主要包含哪些失效模型？简要描述。

故障类型	说明
崩溃性故障	服务器停机，但是在停机之前工作正常
遗漏性故障 接收故障 发送故障	服务器不能响应到来的请求 服务器不能接收到来的消息 服务器不能发送消息
定时故障	服务器的响应在指定的时间间隔之外
响应故障 值故障 状态转换故障	服务器的响应不正确 响应的值错误 服务器偏离了正确的控制流
随意性故障	服务器可能在随意的时间产生随意的响应

56、 分布式系统中冗余的主要方式有哪些?

1. 信息冗余：在数据单元中添加额外的位数据使错乱的位恢复正常 （纠错位）
2. 时间冗余：如果系统出错，一个动作可以再次执行(事务处理) （多次尝试）
3. 物理冗余：通过添加额外的装备或进程使系统作为一个整体来容忍部分组件的失效或故障。 （多配几套设备备用，几个设备一起计算或者进程一起计算）

57、 在分布式系统中如何检测失效?

进程主动往其他进程发送消息检测是否存活。如果得到响应就认为存活，否则就怀疑宕机。

或者被动的等待其他进程的消息。

58、 如何设计可靠的RPC通信机制?

59、 什么是可靠多播？可靠多播存在的问题?

可靠多播就是发送到一个进程组的消息被传递到每个成员

缺点是容易发送方可能被大量的反馈信息淹没

可以改为接收方不对消息反馈，而是丢失了消息才反馈

这样又会导致接收方需要缓存大量旧的信息。

60、 简述两阶段提交协议？ 在参与者失效时如何恢复？ 协作者失效时如何恢复?

Phase 1a: Coordinator sends VOTE-REQUEST to participants (also called a pre-write)

Phase 1b: When participant receives VOTE-REQUEST it returns either VOTE-COMMIT or VOTE-ABORT to coordinator. If it sends VOTE-ABORT, it aborts its local computation

Phase 2a: Coordinator collects all votes; if all are VOTE-COMMIT, it sends GLOBAL-COMMIT to all participants, otherwise it sends GLOBAL-ABORT

Phase 2b: Each participant waits for GLOBAL-COMMIT or GLOBAL-ABORT and handles accordingly.

参与者失效问题：

- INIT：没有问题：参与者还不知道发送的信息；
- READY：参与者正在等待提交commit或者终止abort的信息。恢复后，参与者需要知道它要采取的动作 => 利用日志记录协作者coordinator的决定，或者参考其他参与者的状态
- ABORT: 此时状态是幂等的，重新执行一遍；
- COMMIT: 此时状态也是幂等的，重新执行一遍；

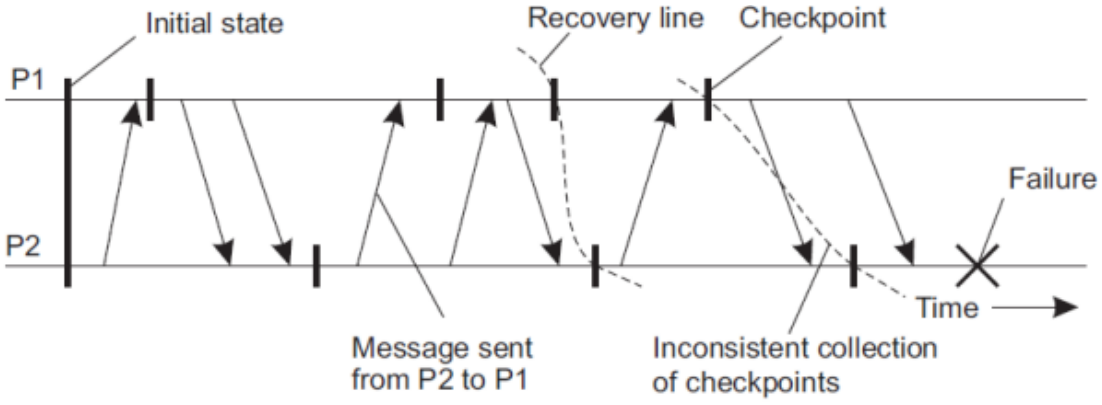
协作者失效

1. （未做决定时）开始2PC时，记录进入WAIT状态，恢复之后重新发送VOTE_REQU EST;
2. 在第二阶段做出决定后，只要记录表决结果就可以了，恢复时重 发这个决定；

61、 分布式系统失效恢复的主要方式?

1. 前向恢复：将整个系统调整到下一个正常状态
2. 向后恢复：回滚到上一个正常状态，设置恢复点recovery point

假定每一个进程都会周期性记录检查点，最近一次的全局一致的检查 点就是恢复线路；



62、 Client-server失效场景及恢复方法?

- **服务器崩溃**

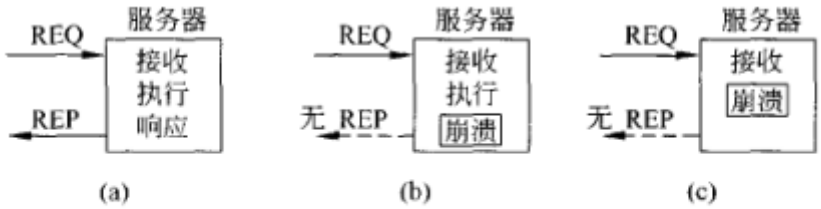


图 8.7 客户-服务器通信中的服务器

(a) 通常情况；(b) 执行之后崩溃；(c) 执行之前崩溃

解决方法：

- **至少一次**：至少执行一次

  航空公司买票：没有响应，多次请求，服务器失效的时候可能已经执行了你的操作，只是传输的时候出现了问题，则出现多次购票的问题

  所以，**写更新不能用**

- **恰好一次**：要么不执行要么只执行一次

  理想的状态，很难达到

完全透明的服务器恢复是不可能的

没有一种客户策略和服务器策略的结合可以在所有可能的事件顺序下正确工作

- **丢失应答信息**

  问题：客户端感知到的是没有接收到应答。但是他不能断定原因是丢失请求，服务器宕机还是丢失响应。

  解决方案：设计服务器时，让其操作都是**幂等**的（idempotent）即重复多次执行与执行一次的结果是相同的；

  ✓ 纯粹的读操作；

  ✓ 严格的写覆盖操作；

  但是实际上很多操作天然就不是幂等的，例如银行事务系统

- **客户端崩溃**

  本质的目的：释放资源（避免资源浪费）、保证客户端和服务器的状态一致。

  服务器还在工作，并且持有资源但是没有客户端需要结果（称为孤儿计算）；

  解决方案：

  - 孤儿消灭：当客户端恢复时杀死孤儿；

    重启服务器，将原来的计算丢弃

  - 再生：把时间分为顺序编号的时期，客户端恢复后，向所有的服务器广播新时期的开始，由服务器杀死与客户端相关的“孤儿”；

    跟服务器广播的时间段不一样，则丢弃原来的计算

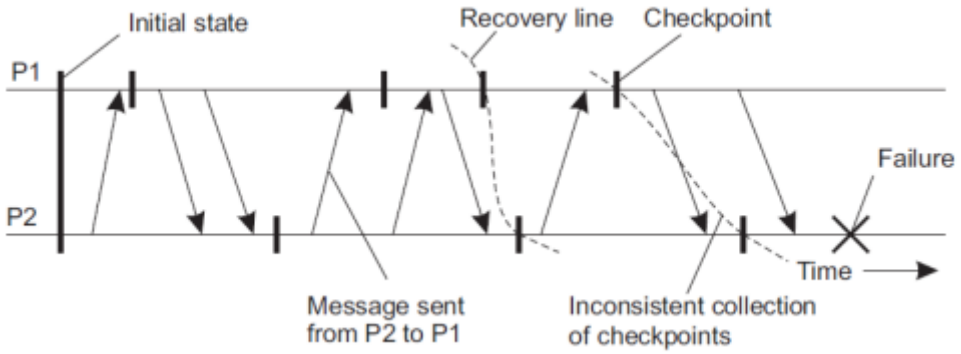
  - 优雅再生：服务器检查远程调用，如果找不到拥有者则杀死计算；

    轮询查询，找不到是谁发出的请求。（额外的代价就是不断的轮询）

  - 到期：时间到期了，终止计算。

63、 什么是检查点方法?

每个进程都周期性把它的状态保存在本地可用的稳定存储中，在进程或者系统失败之后进行恢复，**利用这些局部状态建立一致的全局状态。**



恢复线路：假定每一个进程都会周期性记录检查点，最近一次的全局一致的检查点就是恢复系线路

64、 独立检查点方法？协调检查点方法？

独立的检查点：每个进程的检查点是以一种独立的、不协调的方式来按时记录本地状态，这种分布式特性使得找到一个恢复线路非常困难，可能会导致多米诺效应。 （进程一直回退，直到最初始状态）

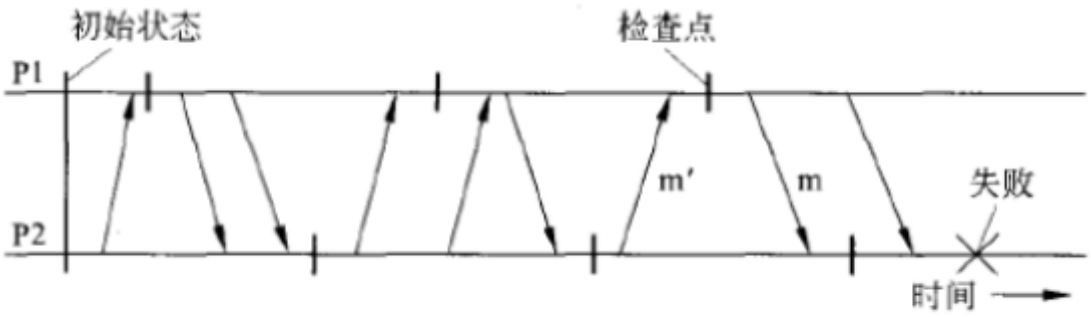


图 8.25 多米诺效应

CPi(m)是进程Pi的第m个检查点

INTi(m)是第m-1个检查点和第m个检查点的间隔。

Pi在INTi(m)间隔中发送消息，

Pj在INTj(n)接收消息，记录依赖INTi(m)->INTj(n)，当打检查点CPj这条依赖会记录这个依赖。

Pi回退到CPi(m-1)，Pj必须回退到CPj(n-1)

协调检查点：所有进程都同步地把他们的状态写道本地稳定存储中。优点：保存的状态自动保持全局一致，避免导致多米诺效应。

1. 协调者多播checkpoint request消息
2. 参与者接收到消息，打检查点，停止发送应用消息，向协作者报告他打了检查点
3. 所有检查点的报告都被协作者确认，协作者广播一个checkpoint done消息以允许所有进程继续。

65、 什么是共识？主要的共识协议有哪些？共识的使用场景有哪些？

定义： 使所有非故障进程就由谁来执行命令达成一致，而且在有限的步骤内就达成一致。

前提： 在一个容错组里面，每一个非故障进程执行的命令以及执行的顺序与其他非故障进程相同；

共识协议分类

失效容错：

准确检测失效：泛洪

最终检测到失效：Pasox Raft ZAB

拜占庭容错：BFT、PBFT、PoW

66、 Paxos的主要过程？Paxos算法的特性？ Paxos的主要变种？

可以解决failure，拜占庭式的失效解决不了。

角色：

- Proposers
  - suggest values
- Acceptors
  - consider the values proposedby proposers
  - render an accept/reject decision
- Learners

过程：

- Single Proposal, Single Acceptor

  one accepter：收集proposals，决定值且告诉其他人
- Single Proposal, Multiple Acceptor

  决策的时候，acceptor根据多数的原则

  decision=value accepted by the majority
- Multiple Proposals，Multiple Acceptors

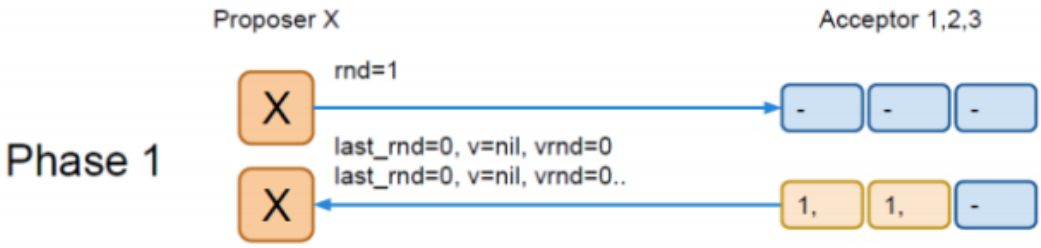
  可能同票

  给提案加上编号，来区分不同的proposal

rnd：每一轮的编号，单调递增；全局唯一（用于区分proposer）

value(v)：acceptor接受的值

vrnd：acceptor接受的v的rnd



当Acceptor收到phase-1的请求时：

- 如果请求中**rnd**比Acceptor的**last_rnd**小，则拒绝请求
- 将请求中的**rnd**保存到本地的**last_rnd**.

  从此这个Acceptor只接受带有这个**last_rnd**的**phase-2**请求。
- 返回应答，带上自己之前的**last_rnd**和之前已接受的**v**.

当Proposer收到Acceptor发回的应答：

- 如果应答中的**last_rnd**大于发出的**rnd**: 退出.
- 从所有应答中选择**vrnd**最大的**v**:

  不能改变(可能)已经确定的值
- 如果所有应答的**v**都是空，可以选择自己要写入**v**.
- 如果应答不够多数派，退出

特性：

完全满足Safety

- Validity：只有一个值被选定
- Agreement：两个正常的结点不会选择不同的值
- Integrity：一个结点最多选一次

最大程度满足Liveness：由于活锁的存在

Paxos种类

- classic paxos
- multi paxos

  1轮RPC，确定一个值
- fast paxos

  没冲突，1轮RPC确定一个值

  有冲突，2轮RPC确定一个值

67、 Raft共识的主要特点是什么?

safety：每个term最多允许一个winner

- server只能投一票
- 赢得election必须获得大多数的投票

liveness：满足，some candidate 最终会获胜（通过时间限制来控制）

68、 NFS文件系统的主要架构?

客户-服务器体系结构

NFS每个文件服务器都提供其本地文件系统的一个标准化视图。每个NFS服务器都支持相同的模型;底层模型是远程文件访问模型;

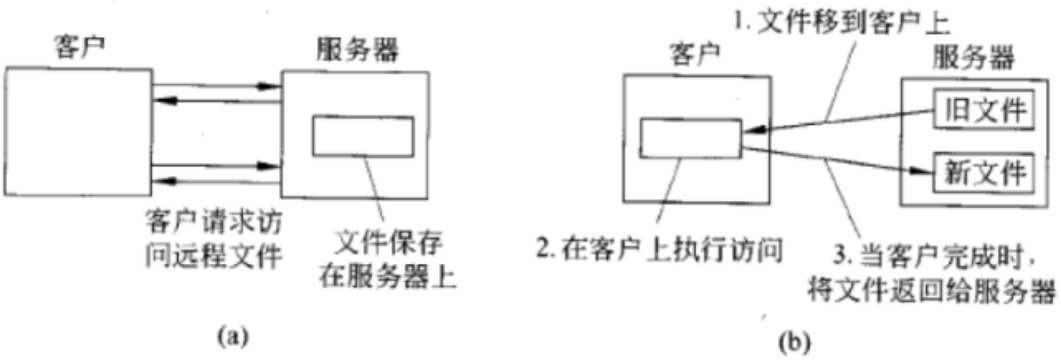


图 11.1

(a) 远程访问模型；(b) 上载/下载模型

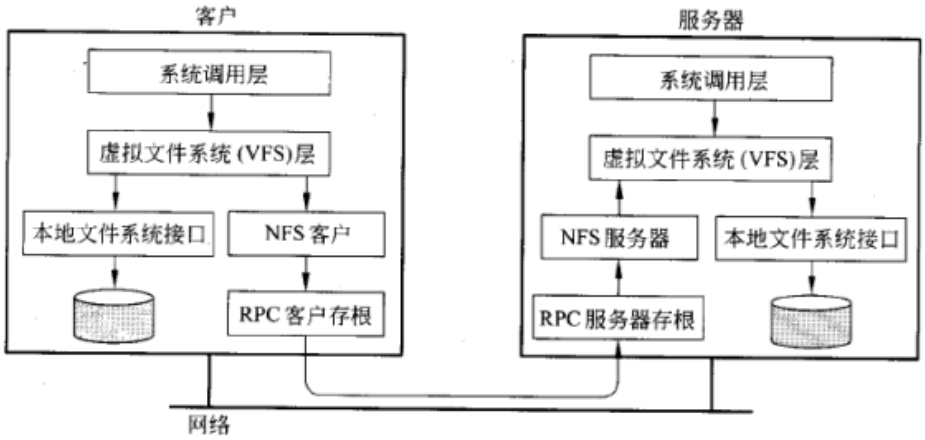


图 11.2 用于 UNIX 系统的基本 NFS 体系结构

NFS采用VFS（Virtual File system）实现，VFS是不同文件系统接口事实上的标准，VFS提供了系统结构，**屏蔽了访问本地和远程文件的差异性。**

NFS支持硬链接和符号链接，不允许嵌套挂载

无状态的NFS： 实现简单， 服务器崩溃后，不需要恢复阶段；无法锁定访问文件；

有状态的NFS： 需要服务器维护关于客户端的大量信息；

- **基于集群的分布式文件系统**
  - 加快文件访问速度，应用文件分片划分技术使文件可以并行访问。
  - 将文件分成较大的数据块如64MB，将数据块分布、复制到多个物理机器上。

    master（目录服务，记录文件对应的chunk server的位置—哈希表）、chunk server（存具体的数据内容）

    chunk server更新：告诉master，做过更新（数据块可能移动了）

    master上的视图并不是实时的：与chunk server的内容并不一定实时同步，否则代价很大。

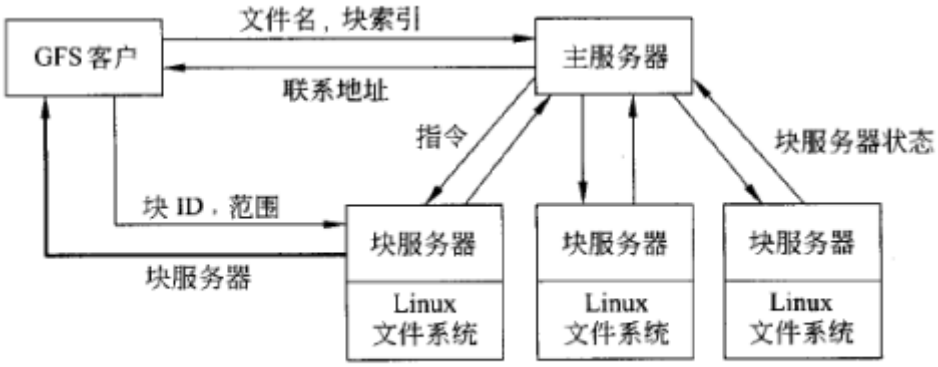


图 11.5 Google 服务器群集的组织结构

对称式体系结构

基于对等技术的完全堆成的体系结构

DHT：每个文件要存到哪些节点上

69、 GFS文件系统的主要特点?

google file system

每个GFS集群都由一个master服务器和多个chunk服务器组成。GFS文件被分成多个块，将这些块分布在chunk服务器上。块由一个关联的标识符，允许chunk服务器找到他。

1. 主服务器只是在内存中维护了一张(文件名，块服务器)的映射表 = > 最小化IO，以日志的形式保存对数据的更新操作，当日志过大时将创建检查点，存储主存数据;
2. 大量的实际工作是块服务器完成的

   当客户需要访问数据时，他会联系主服务器，查明哪些chunk服务器保存有该数据。之后客户只会与该chunk服务器通信。
3. 文件采用主-备份模式复制块，主服务器会避开循环;

   当客户执行更新操作时，他会联系最近的保存该数据的chunk服务器，并把它的更新推送给服务器。这个服务器将更新推送给下一个chunk服务器，传播所有的更新，客户联系master服务器，给更新操作分配一个序列号，传给备份。

   chunk可能被某个客户端更改，其他的备份的chunk也要保持更新，这个过程不是master发起的，而是谁被更改了谁去发起更新。
4. 上述两个特点决定了GFS主服务器不会构成瓶颈，为系统带来重要的可伸缩性，单个主服务器可以控制数百个块服务器;

70、 什么是文件系统语义模型？主要的语义模型包括哪些？简要描述其特征。

文件共享语义：当需要处理分布式文件系统时，我们需要考虑并行读写的操作顺序和期望的语义（处理好一致性）

71、 拜占庭容错的基本思想及基本过程?

基本思想是通过构造有限状态机的集合来部署主动复制，并且这个集合中具有无故障的进程以相同的顺序执行操作。

Byzantine故障解决方案

- 为了在异步环境中进行Byzantine容错，服务器组必须包含至少 3k+1 个进程；
- 难点：确保无故障的进程**以相同的顺序**执行所有的操作。
- 简单解决方案：指定一个协调器，它通过简单地给每个请求附加一个序号来序列化所有的操作。问题转嫁到协调器身上；

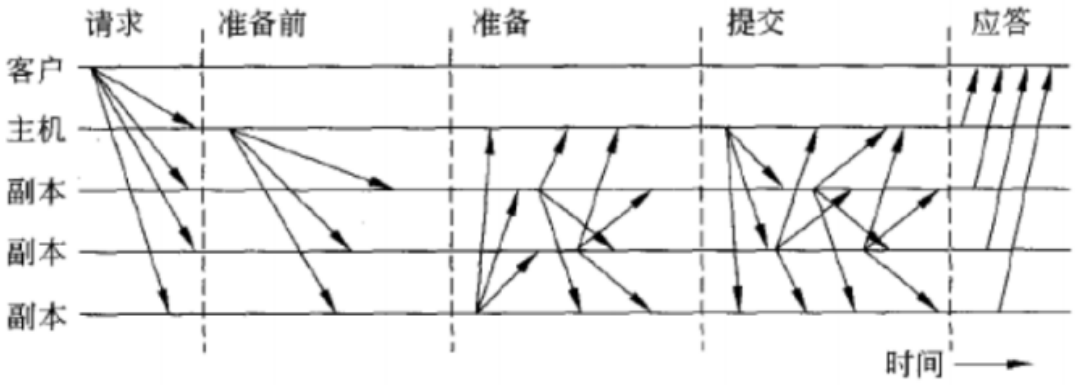


图 11.26　Byzantine 容错性中的不同阶段

进程会形成一系列视图， 每个视图都会对无故障的进程达成一致意见，并且在当前主机看起来失败时更新视图。

该协议的关键：可以正确地对请求排序；

72、 P2P系统中用于提高系统可用性的方案？以及方案的主要特点。

问题：P2P系统的节点的不可用性非常高，简单的复制文件已不能保证可用性。

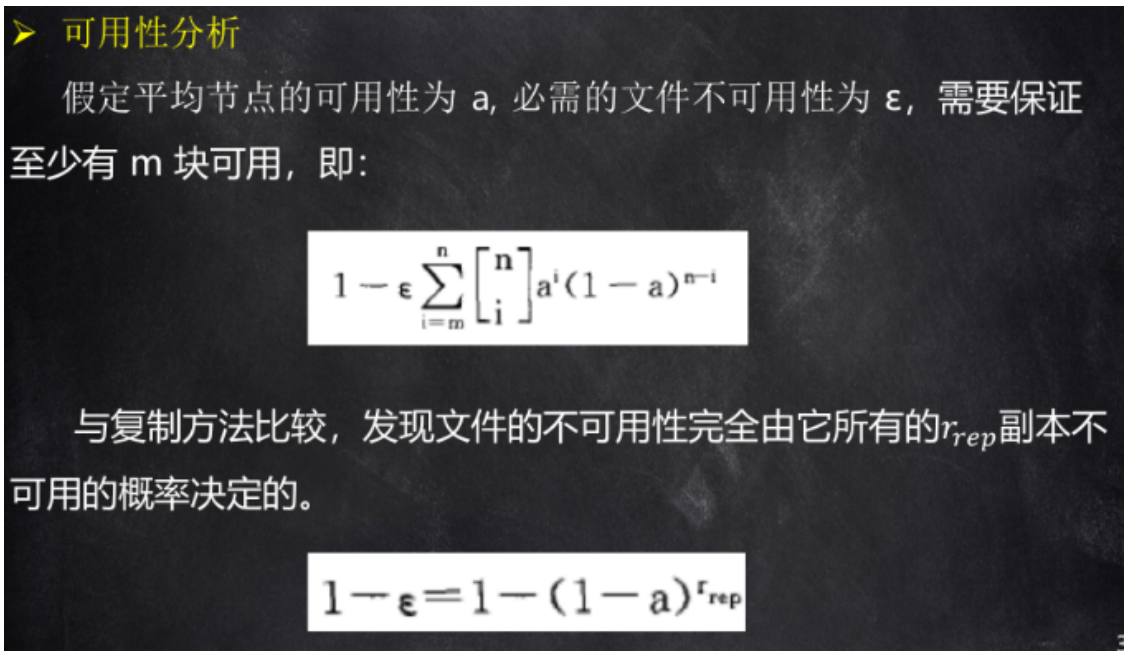
冗余性方案：

1. 复制：通过放置多个副本提高数据的冗余度从而提高可用性。
2. 擦除编码（erasure　coding）：通过把一个文件分成m块，随后把它记录到n大于m块中，任何m个编码块的集合都足以用于重构造原始文件：

   冗余性因子

$$r_{ce} = \frac{n}{m}$$

   擦除编码



73、 在分布式文件系统中主要关注的问题包括哪些？并分别给出一些解决问题的方案。

- 命名

  对于一个大型分布式系统，当需要提供对文件的共享访问时，至少要有一个全局名称空间。

  全局名称空间服务(GNS);

  把现有文件系统集成进单个全局名称空间中，只使用用户级解决方案

- 同步

  当多进程同时操作文件时，出现问题。

  - 文件共享语义：当需要处理分布式文件系统时，我们需要考虑并行读写的操作顺序和期望的语义（处理好一致性）

    对打开的文件所做的改动最初只对修改文件的进程可见，只有当文件关闭时，这些改动才对其他进程可见，即会话语义;

    操作文件的进程看得见，其他进程看不见。—很多场景采用的。

  - 文件锁定：
  - 共享预约：包含客户打开文件时指定的访问类型，以及服务器应该拒绝的其他客户的访问类型。（相当于自己决定允许别人进行什么操作）

- 一致性
- 安全性
- 容错性
- 可用性

74、 分布式机器学习的参数服务器如何理解？

75、 如何保障参数服务器状态的一致性？

76、 MapReduce计算泛型的主要流程是什么?

Map/Reduce Operation

优势:

1. 强容错能力——冗余方式
2. 适用于大规模数据集的场景

缺点:

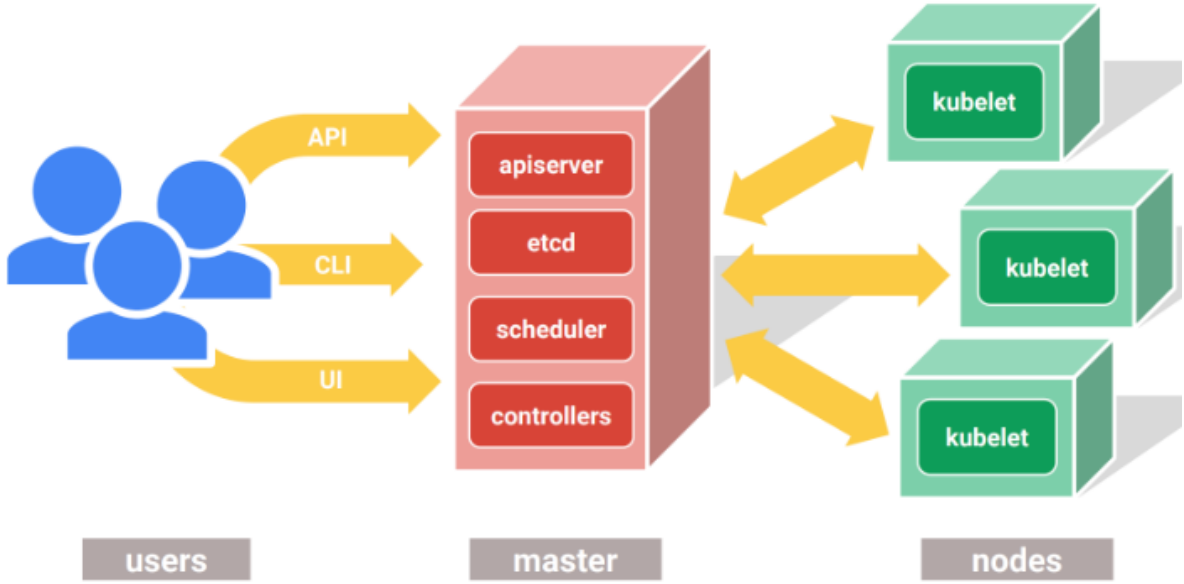
1. higher overhead
2. lower raw performance

尤其是对于硬件来说，屏蔽了底下硬件的特点，没办法利用底层应用的优势。

77、 MapReduce计算泛型的主要特点是什么?

78、 Kubernetes的主要组成是什么?主要功能又是什么?Kubernetes中的service如何理解?

http://dockone.io/article/932



apiserver：接口，可以和集群交互

etcd：容器的信息，集群元信息

scheduler：分配调度容器

controllers：控制容器，比如update kill

kubelet：基本控制容器，如pod管理，开关容器，生命周期等

kuproxy：代理

API：通过接口和集群交互

CLI：命令行工具，和集群交互

UI：用户界面，直接拖拽

service是对运行实例比如pods进行外包装，分配IP地址让外部访问