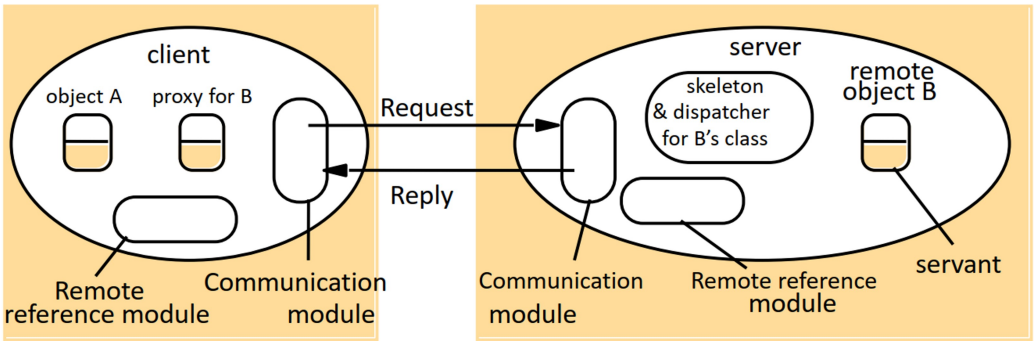
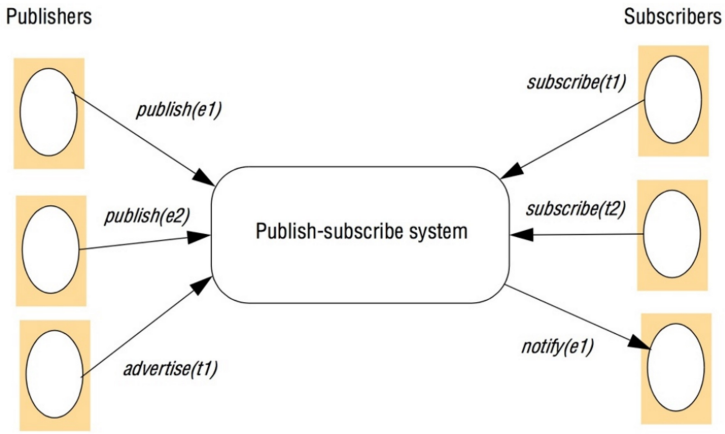
分布式题库

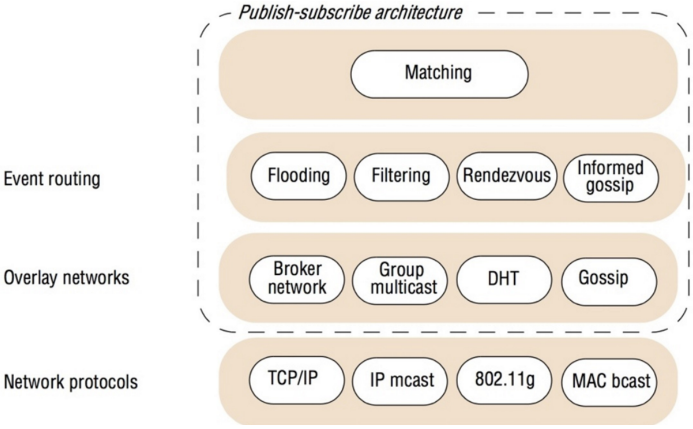
2021年5月30日 16:29

- 第一章：分布式的特征
  - Flynn taxonomy 是什么？对其中的分类各举一些例子。
    - 通过指令和数据的对应关系来划分计算机架构，包括：SISD（如传统的单处理器）、SIMD（如GPU）、MISD（较为少见，如异构系统操作同一个数据）、MIMD（如多核处理器和分布式系统）
  - scale-down、scale-up、scale-out、scale-in分别是什么?
    - scale-down、scale-up：增加
    - scale-out、scale-in
  - 分布式系统的定义是什么?
    - 一个由多台计算机构成的系统，这些计算机由网络连接并通过消息传递来协调行为
  - 列出两个构建分布式系统的挑战
    - 异构性（heterogeneity）、安全性、可扩展性、对于failure的应对（detection、tolerate、recover）、consistency、transparency
  - 列举三个分布式系统中追求的transparency
    - location transparency：在不知道资源的具体位置（如IP地址）的情况下获取资源
    - replication transparency：在使用者不知道副本存在的情况下使用某个资源的多个副本来响应请求
    - failure transparency：在软件或硬件出现failure的情况下依旧完成任务
    - mobility transparency：在资源或client移动时不影响用户使用
    - scaling transparency：当系统和应用的规模改变时系统架构和应用算法都不必更改
- 第二章：系统模型
  - system model中的one-tier、two-tier和three tier是指?
    - one-tier：将presentation layer、application logic layer和data layer放在一起
    - two-tier：将presentation tier放在client，application logic layer和data layer放在一起
    - 将presentation tier、application logic layer和data layer分开，其中application logic layer和data layer通过middleware相连（包括JDBC、RPC等）
  - 分布式系统设计的metrics有哪些？大致解释一下。
    - performance：响应速度、吞吐量、负载均衡
    - QoS：deadline property（soft deadline或hard deadline）、adaptability（可适应系统配置的更新）
    - Dependability：正确性、容错、安全
  - dropping message error又被称为omission failure，它被分为哪三种?
    - send-omission：进程结束了send操作，但是message没被放在sending buffer里
    - omission：被放在sending buffer的message没发到receiving buffer里
    - receive-omission：message被发到了receiving buffer里，但相应的进程没有接收
  - 在class of failure中，fail-stop和crash的区别是什么？设计一个能够实现fail-stop的场景
    - fail-stop指当一个进程crash时另一个进程能够确切地感知到。
    - 场景：在同步系统中，可以设置一个timeout，如果一个进程p1发出message后的timeout时间内接收者p2都没有回复，则说明p2 crash了。
  - Byzantine failure是什么?
    - 它包含了所有的种类的failure，即所有进程和channel能产生各种行为，包括：进程收到信息后不回复或回复错误的信息等，这是failure的上界，表示最极端悲观的情况
  - 在所有failure中，是受channel影响而产生的failure有哪些?
    - Omission failure、Arbitrary （Byzantine） failure
  - timing failure包括哪些?
    - clock failure：进程的本地时钟超过了rate of draft from real time的bound
    - performance failure of process：Process exceeds the bounds on the interval between two steps
    - performance failure of channel：消息传递时间超过了bound
- 第三章：进程间通信
  - protocol的定义是什么？protocol和service有什么区别?
    - 每一层封装的功能

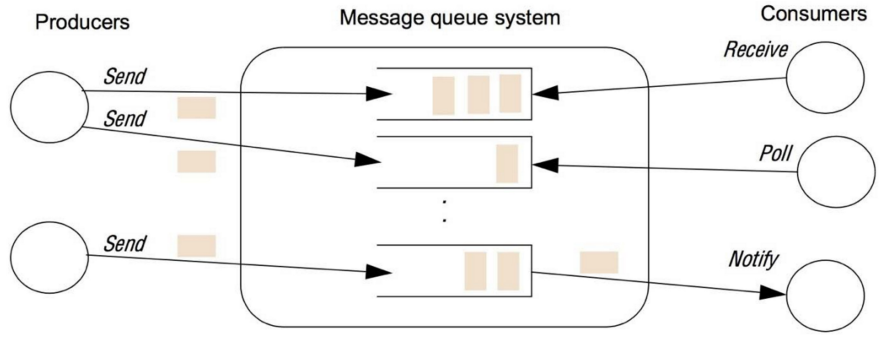
        - 区别：service是提供给高层的一些primitive，而protocol定义了这些primitive的实现
    - multicast有什么特点？和重复发送数据相比有什么好处？
        - multicast使得发送进程发送的一条消息能到达group内每一个进程
        - 和重复发送数据相比，它的性能更好
- 第四章：分布式对象和远程调用
    - 在分布式系统中，应用层和OS之间的层叫做什么？
        - Middleware layer
    - 一个object包括哪两个部分？
        - data（attribute）和operation（method）
    - 介绍一下RMI中的各个component，并画出两个进程通过RMI通信的大致结构
        - proxy：对于每一个remote object在client端都有一个相应的proxy，proxy伪装成local object，使得invoker能直接像调用本地对象那样调用它，而不用了解它的内部实现
        - remote object module：将local object reference（proxy）翻译成remote object reference
        - communication module：实现request-reply协议
        - dispatcher：从communication module中收到请求，选择skeleton中合适的method
        - skeleton：解析请求中的参数，调用servant中相应的方法，然后返回处理结果
        - servant：位于server进程中，是remote object的实际body，负责实际处理关于远程对象请求
        - 

    - 远程调用中有哪些invocation semantic?
        - Maybe、at least once、at most once、exactly once
- 第五章：间接通信
    - space coupling和time coupling分别是什么？再写出（time coupling和非time coupling）*（space coupling和非space coupling）的样例表。
        - space coupling：发送者要知道接收者的具体地址
        - time coupling：发送者发送消息时，接收者必须在场
        - 时空都耦合：massage passing、remote invocation
        - 只有空间耦合：email
        - 只有时间耦合：IP multicast
        - 都不耦合：大多数的间接通信范式，如消息队列
    - open group和close group分别是什么?
        - open group指group外的人能联系到group内的，反之为close group
    - 在分布式系统中，通信的ordering有哪些？简单解释一下
        - FIFO：所有消息都按照发出顺序被收到
        - causal：如果消息m2是作为消息m1的结果而被发出（如一个进程看见了m1然后发出了m2），那么所有人都要先看见m1
        - total：所有人收到消息的顺序都一样
    - 画出publish-subscribe paradigm和 publish-subscribe systems 架构图

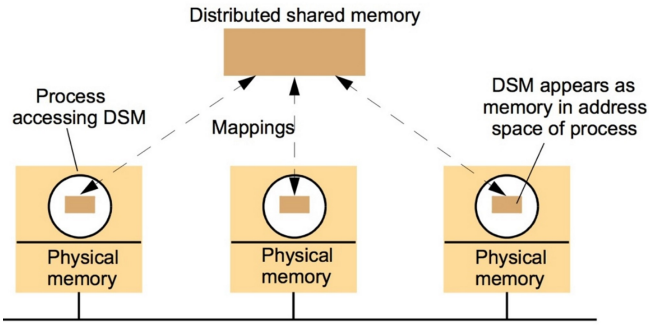




- pub-sub系统中有哪些routing 方法?
  - flooding：pub时发给所有节点，然后在sub端自行选择是否match；也可以在sub时发给所有节点，然后在pub端自行选择是否match，如果match就加入pub-list
  - filtering：pub维护着sub名单，对应着每个routing方向要发送的订阅者名单，每次只给这些人发送
  - rendezvous：pub/sub时随便发给一个节点，如果这个节点是负责这个subscription的rendezvous节点，则把这个节点加入订阅名单或notify相应的订阅者，否则转发
- 画出MQ的图



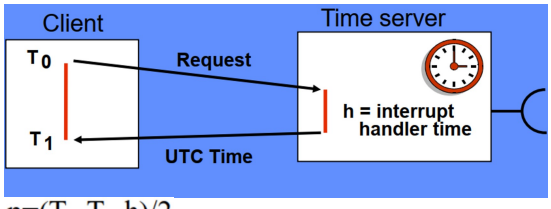
- 画出distributed shared memory的图



- pub/sub、MQ、DSM中，space-uncoupled的有哪些？肯定 time-uncoupled的有哪些？它们的style of service分别是什么?
  - space-uncoupled：pub/sub、MQ、DSM
  - 肯定time-uncoupled：MQ、DSM
  - style of service：pub/sub：communication-based、MQ：communication-based、DSM：state-based

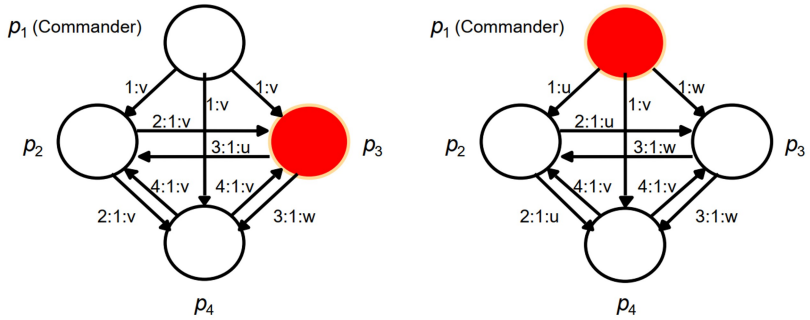
- 第六章：OS Support

- program、process、thread的区别是什么?
  - program是一个静态的可运行程序
  - process是program的一个运行实例，等于若干个线程加上运行环境
  - thread是轻量级进程，是OS对一个行为的抽象
- 假设对于每个请求需要2ms processing + 8ms 磁盘I/O，分别计算单线程+无磁盘缓存、双线程+无磁盘缓存、双线程+有命中率为75%的缓存下的吞吐量（每秒并发数）
  - 单线程、无磁盘缓存：1000/(2+8)=100 个/s
  - 双线程、无磁盘缓存：1000/(2*0+8)=125 个/s，因为processing可以在对上一个请求IO时进行
  - 双线程、有命中率为75%的缓存：1000/(2*0+ 8*25%)=500 个/s
    - 但考虑到利用和维护缓存的代价，最好是1000/2.5=400 个/s
- 写出几个多线程对象服务器架构并说明其优缺点
  - worker pool
    - 大G的课件中默认其为固定大小的线程池
    - 可以调整优先级，但是不灵活
  - thread-per-request
    - 无请求队列的queuing，实时性好一些，但是创建消毁的开销大
  - thread-per-connection/object
    - 开销低，但会导致负载失衡

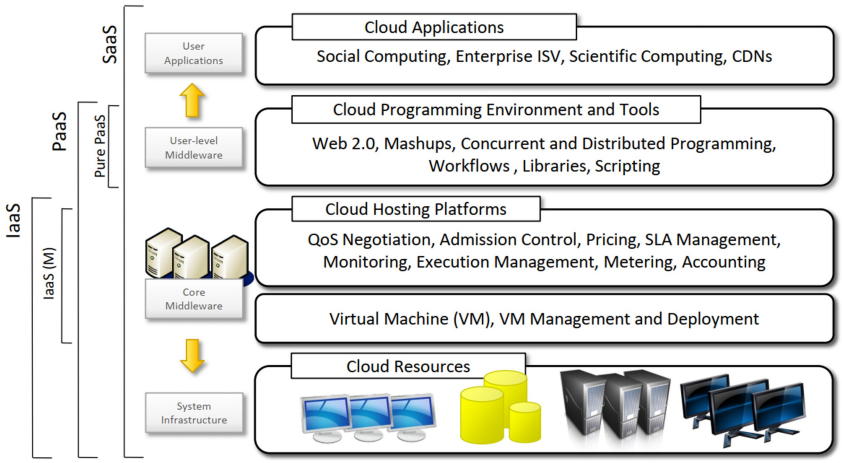
- 第七章：Time global state
  - 列出time service在分布式系统中的三个作用
    - 测试系统中不同组件的时延
    - 同步不同的数据流，如声音和视频
    - 建立起事件顺序（event ordering）
    - 为了执行分布式的数据库transaction，如使其满足serializability隔离等级
  - clock skew和clock drift分别是什么?
    - clock skew是指两个clock时间不同
    - clock drift是指在标准完美的时钟跑过单位时间内，一个时钟的读数变化差异与之不同
  - UTC是什么？是怎么传播的?
    - Universal Coordinated Time（因为是从法语中来的，所以定语后置），是基于原子钟的国际时间
    - UTC是用GPS来广播的，然后通过NTP逐层传播
  - 时钟同步分为External synchronization和Internal synchronization，分别表示什么?
    - External synchronization指通过权威时间来同步，在每个时刻，clock的时间和权威时间之差都在一定范围内
    - Internal synchronization指各时钟互相同步，彼此之差在一定范围内
  - 如果有两个drift=R的时钟，那么为了能把它们的时差控制在D秒内，它们最久需要多久同步一次?
    - 由于它们的最大时差变化为 2R，所以同步间隔最小应该是：D/2R
  - 对于Internal synchronization系统，假设已知两机间消息传递的时延bound为[MIN, MAX]，那么当一个带有时间戳time=t的消息从p1到达p2时，最好将p2的时间同步为?
    - t+(MIN+MAX)/2，这样的话误差上界最小
  - 描述一下Cristian's 算法
    - 

    - $p=(T_1-T_0-h)/2$
  - 描述一下Berkeley算法
    - 找一个master作为协调者，将master的时间发给slave，然后slave返回自己的时间和master的时间的差值。
    - master忽略异常的延时，然后对各个节点的时间求平均值
    - master将每一个节点应该做的时间调整发给相应的节点
    - 如果master挂了就挑选其他的master
  - 描述一下Lamport Clock?
    - Lamport Clock是一种单增的counter
    - 每个进程都有一个Lamport Clock的timestamping

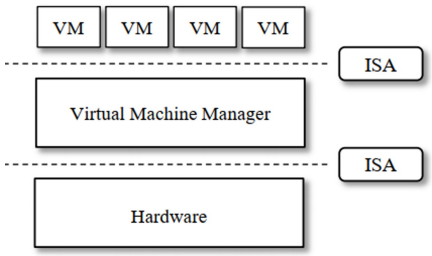
        - 如果一个进程需要给一个event赋个时间戳，则先给这个Lamport Clock加一个值
        - 如果进程q收到一条带有时间戳 t 的消息，则 Lq=max(Lq, t)+t2，其中t2是一个增量
    - 怎么基于逻辑时钟实现totally ordered logical clock?
        - 方法一：将clock扩展成 (T(a), pid)，如果时间一样就比较pid
        - 方法二：使用 Vector Clock
            - 一个clock由N个元素构成，其中N为系统中的进程数
            - 如果一个进程要给一个event赋个时间戳，则先给这个Lamport Clock的相应的分量加一个值
            - 只有一个时间戳的各个分量都不小于且某个分量大于另一个时间戳时才算二者构成了event ordering
- 第八章：Coordination and Agreement
    - 写出分布式互斥的三个要求
        - ME1：相同时间内，最多只能有一个进程进入critical section
        - ME2：enter和exit请求最终都会被满足
        - ME3(可选的)：以causality order来满足进入的请求
    - 列出并描述三个分布式互斥的实现算法
        - centralised service
            - 进程向server发出token请求，只有当critical section是free时才会授予，否则将请求入队
            - 问题：单点失败、单点性能瓶颈
        - Ring-based 算法
            - 环大时时延很大、对网络带宽消耗大
        - Ricart-Agrawala算法
            - 使用Lamport时钟，保证了causality order)
            - enter：
                - 先将自己的state设置成wanted
                - 多播请求给所有进程
                - 如果收到了所有的回复，则将state设为held
            - receive request：
                - 如果state=held，或者 （state=wanted 并且 (T, p_j)<(T_i, p_i)）则将请求入队
                - 否则回复
            - exit：
                - state=released
                - 回复队列中的所有请求
    - 写出分布式leader electio的两个要求
        - LE1：每个进程都知道某个进程是leader
        - LE2：所有进程都参与了选举并能最终选出leader
    - 描述一个建立在reliable link基础上的分布式leader election算法
        - Chang&Roberts算法：
            - 所有的节点构成了一个单向环
            - 每个节点都有participant和non-participant两种身份，一开始都是non-participant
            - stage1：election
                - 发起者（initiator）变成了participant，将其UID传给了邻居
                - non-participant进程收到election消息后，转发带有max(received UID, own UID)的election消息，变成participant
            - stage2：elected
                - 当participant收到UID是它的election消息，则变成为leader和non-participant，然后转发带有其UID的elected message
                - 当participant收到UID不是它的elected消息，则记录leader的UID，变成non-participant，并转发消息
    - Chang&Roberts最多需要几个round-trip才能收敛?
        - 3个。假设initiator进程P1的前一个进程P0是最终被选出的leader，则election message会被传两轮才能让P0知道自己是leader，从而开始为期一轮的elected message阶段
    - 列出consensus的三个requirement
        - termination：算法能终止到每个正确的进程都有decision value
        - agreement：任意两个正确的进程的decision都相同
        - integrity：如果所有正确的的进程的提出了相同的值，则所有正确的进程都要选择那个decision

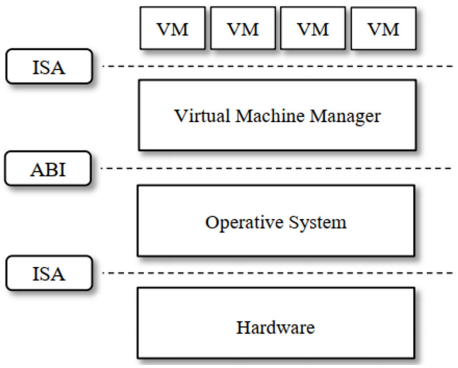
- 设计一个用于同步系统并且能确保fault-tolerance的共识算法
  - 一开始所有的进程都提出一个值
  - 每个进程都维护在round r时它已知的值，然后发送它之前没有发送过的值
  - 每个进程都把收到的新值加入已知的值
  - 在round f+1时，每个进程选择最小的已知的值作为decision，其中f表示希望最多接受f个进程crash（使用timeout）
- Byzantine将军问题是什么?
  - 进程可能出现Byzantine failure
  - 所有进程要对一个decision达成共识
  - 有一个进程是commander，给其他进程发送命令
- 怎么证明他同步系统中当三个将军中有一个failure时无法保证达成共识?
  - 有两种failure场景：commander fail 或 普通将军failure
  - 非failure的将军无法区分这两种情况
  - 所以，如果commander出现failure，则两个普通的将军可能有不同的决定
- 画出同步系统中当四个将军中有一个failure时的可能情况
  - 

  - 如果将军非fault，则可以通过majority得出一致结论
  - 如果将军是fault，则无法通过majority得出结论，所以不会采取行动
- 在有N个节点的异步系统中最多能容忍几个Byzantine fault的节点从而保证达成共识?
  - 0个。

- 第九章：cloud
  - 根据用户划分，cloud可以分成哪三类?
    - public、private、hybrid
  - SaaS、PaaS、IaaS分别是什么？举几个例子?
    - SaaS：software as a service：for end user app：Google Document
    - PaaS：platform aaS：runtime environment for app：Hadoop
    - IaaS：vitualized server（存储和网络）：EC2
  - 画出云计算的架构
    - IaaS：system infrastructure、Core Middleware
    - PaaS：IaaS、User-level Middleware
    - SaaS：PaaS、User Application
    - 

  - Virtualization Reference Model包括哪三层?
    - Guest、Virtualization layer、Host
  - virtualization分为哪四层?
    - 应用、编程语言、OS、硬件 这四个level

- sensitive 指令可以分为哪两种?
  - behavior-sensitive：读privileged state
  - control-sensitive：写privileged state
- 分别画出type-1和type-2的hypervisor
  - 

  - 

- hypervisor的三条theorem是指?
  - 如果sensitive指令都是privileged指令的子集，则可以构建VMM（VM Manager）
  - 如果一台计算机是可虚拟化的并且不需任何timing dependency就能在其上构建VMM，则它是可递归虚拟化的
  - 如果一台计算机中的user-sensitive指令都是privileged指令的一部分，则可在其上构建hybrid VMM
- 硬件虚拟化方式包括哪些?
  - 硬件协助虚拟化：在完全隔离的环境下跑guest OS
  - Full 虚拟化：在VM上跑OS。
    - 如：VMware。其中桌面版是type II，server版是type I
  - paravirtualization：非透明，guest和VMM需要co-design，比较简单
    - 如：Xen
  - partial虚拟化：不允许guest OS在完全隔离的环境下执行，只支持部分的OS功能

- 其他（懂的都懂）
  - 证明在有数字签名时，三个节点的拜占庭将军问题能保证达成共识
  - 比较一下用TCP这种有连接的协议和UDP这种无连接的协议实现一些算法的不同
  - type-1和type-2的hypervisor有什么不同?