- introduction
    - 一组自治的计算机组件，对用户来说表现成一个一致的系统
        - 微处理器更经济；
        - 速度快；
        - 物理分布式；
        - 可信度高；
        - 便于横向拓展
    - Transparency: Data Access; Resource Location; Migration; Relocation; Replication; Concurrency(Resource shared); Failure
    - Openness (Interact with others)
    - Allow set up policies (cache policy, level of consistency, QoS, level of secrecy)
    - Scalability: 节点或用户数量; 地理位置; 管理域
    - Distributed Transaction:
        - ACID: 原子性 Atomicity, 一致性 Consistency, 隔离性 Isolation, 持久性 Durability
- Architecture
    - 中心化 Centralized: Clients and Servers
        - Multiple Clients / Single Server
            - bottleneck, single point of failure, scaling difficult
        - Multiple CLients / Multiple Servers
            - Web Applets: 运行于客户端的程序组件
            - Traditional three-layered
                - User-interface layer, Processing layer, Data layer
    - 去中心化 Decentralized:
        - 结构化P2P：节点按特定结构组织，logical ring, hypercube, hash (每个节点提供特定ID的服务)
        - 非结构化P2P：节点随意选择邻居，two nodes are linked with probability
            - look up information:
                - Flooding: 向所有领取发送查询请求，若有结果则返回，若没有则重复以上步骤
                - Random walk: 随机选择邻居，若有结果则返回，若没有则重复以上步骤
        - 混合P2P：部分提供特定功能的节点按特定结构组织
            - BitTorrent: Trackers

- Process
    - An execution stream in the context of a process state
        - process: 动态的代码和数据实例
        - program: 静态的代码和数据
        - execution stream: piece of codes, sequential sequence of instructions
        - process state: everything running code can affect or be affected by
    - Three modes
        - Running: on the CPU
        - Ready: wait CPU
        - Blocked: wait for I/O or synchronization
    - Context
        - minimal collection of values stored in the register of a processor
        - Context Switching
            - Thread Context Switching: 独立于操作系统
            - Process Context Switching: trap，转为内核态
    - Thread
        - Share same memory address
        - implementation
            - User-Level Thread

用户级线程是指不需要内核支持而在用户程序中实现的线 程，它的内核的切换是由用户态程序自己控制内核的切 换，不需要内核的干涉。但是它不能像内核级线程一样更 好的运用多核CPU。

                - less trap to the kernel
                - thread interact with process instead of kernel
            - Kernel-Level Thread: Lightweight Process (LWP) in Kernel
                - OP block thread: kernel schedule another thread in the same process
                - Handle event: kernel schedule the threads with the event
                - Loss Efficiency
        - Advantage
            - client: hide network latencies; multiple RPCs to other services, speed up
            - server: performance, start thread is much cheaper; better structure
    - VirtualMachine
        - ProcessVM: 程序被编译成中间代码，JVM
        - VM Monitor: 用于模仿硬件的软件层，VMware, KVM, Xen
            - binary translation
            - sensitive instruction (system call) replace with call to VM Monitor

- Multithreaded Clients
    - Browser
- Multithreaded Servers
    - dispatcher / workers
- Stateless Service
    - never keep track of information about the client
    - independent; state inconsistence are reduce; loss performance
- Stateful Service
- Code Migration
    - code segment
    - data segment: contains state
    - execution state: contains context of thread executing the code
    - weak migration
        - move only code and data segment, reboot execution
    - strong migration
        - 复制进程 (正在执行的进程停下来，移动后再恢复)
        - 完全clone，设置为相同的execution state
    - local resource
        - type
            - fixed (硬件)
            - fastened: move with high cost
            - unattached (缓存)
        - object-to-resource binding
            - by identifier: require specific instance of resource (特定数据库), can MV
            - by value: require the value of a resource (缓存实体集合), can CP
            - by type: require a type of resource (显示器) can Re-bind
    - 在异构系统中的迁移
        - 目标计算机无法执行代码；进程/线程上下文的定义与操作系统或运行系统有关
        - 解释型语言；虚拟机；

- Communication
    - Layered Protocols
        - Low-Level: Physical Layer, Data Link Layer, Network Layer
        - Transport: TCP && UDP, IP多播
        - Middleware: set of communication protocols, naming protocol, security protocol

- Type
    - Transient communication (非持久化通信)
        - 消息的发送双方，必须同时在线，否则消息被丢弃
    - Persistent communication (持久化通信)
        - 对消息做持久化，直到发送到目标为止
    - 异步通信
        - 消息发送后继续执行
    - 同步通信
        - 消息发送后，阻塞直到同步点
            - At request submission: 发送消息提交到传输服务
            - At request delivery: 消息被接收者成功接收
            - After request process: 消息的处理结果告知给发送者
    - Client / Server: Transient Synchronous
    - Messaging: Persistent Asynchronous
- RPC: Remote Procedure Call
    - operation
    - 参数传递
        - Client和Server对数据有不同的表示方式 (byte order, encoding)
    - Failure
        - Client unable to locate the server (服务器down或版本不一致)
            - Specific Error Code as RPC Return
        - Lost Request Message
            - 设置定时器，超时重发；太多消息丢失 => unable locate server
        - Lost Reply Message
            - 设置定时器，超时重发 / Reply ACK
            - Procedure会被多次执行(幂等？)，给请求添加序号
        - Server Crash
            - Receive => (Crash?) => Execute => (Crash?)
            - 等待服务器重启，重发请求 (至少一次执行)
            - 立即放弃 (最多一次执行)
            - do nothing
        - Client Crash

            - orphan(孤儿): computation is running and no parent is wait result
            - extermination(消除): Log before send RPC. Kill orphan by log when reboot.
                - 额外的磁盘读写，孙子orphan的消除，网络不通
            - reincarnation(再生)
                - 广播epochs, kill computation when recv new epoch
            - gentle reincarnation
                - kill computation not has owner
            - expiration
                - each RPC has Expiration T
    - Interface Definition Language (IDL)
        - generate client and server stub
        - can not send pointer or system specific data(LOCK, fd, socket)
    - Dynamic binding
        - when server start, send msg to binder
            - Name + Version + ID + Handle(address)
        - when client send RPC
            - send LOOKUP (Name + Version) msg to binder
            - binder response with (ID + Handle)
        - advantage
            - 灵活性
            - 多个服务提供相同的接口
                - 负载均衡
                - binder检查service是否可用，容错性
                - auth：server告知binder可被哪些user调用
            - binder检查版本是否一致
        - disadvantage
            - cost
            - binder become bottleneck
- Message-Based Communication
    - 同步
        - sender阻塞直到msg被存储到receiver buffer
    - 异步

        - msg被存储到sender buffer
    - Transient
    - Persistent
    - Message Queue System
        - sender and receiver both have queues
        - Message Broker
    - stream-oriented communication
        - continuous media: audio, video
        - time guarantee (end-to-end delay):
            - asynchronous
            - synchronous (max end-to-end delay)
            - isochronous (range end-to-end delay)
        - stream
            - unidirectional (单向的)
            - single source / multi sink
            - Quality of Service (QoS)
                - Bit rate
                - Delay util session setup
                - End-to-End Delay
                - jitter (use buffer to reduce)
                - packet loss
            - complex stream synchronization (音频和视频复合流)
    - multicast communication
    - Update Replication
        - Anti-entropy
            - Node P random choose Node Q, (push | pull | push-pull) updates，log(N) rounds
        - Gossiping
            - Server S who has update contact other servers, if one server already has update, stop with probability 1/k
        - Remove Value
            - Removal => special Update
- Naming
    - Kinds

        - Human-friendly name: Hostname
        - Address: IP
        - Identifier: MacAddress
            - identifier => entity <= 1
            - entity => identifier <= 1
            - identifier => same entity
    - Name => Access point
        - Broadcasting: request the entity response with address
            - fallback: only in LAN, every one listen
        - Forward pointer: A => B, leave a proxy at A, forward request to B
            - fallback: 链会很长，且容易断
        - Home-based
            - Home keep track of where entity is
            - Client contact home first, get entity address
            - fallback: home必需保证一直可用，entity永久移动带来不必要的开销，client与entity相邻
    - DHT
    - Hierarchical Location Service (HLS)
        - Leaf: address of entity
        - Intermediate node: pointer to child which has entity address
        - Root know all entity
        - Lookup
            - start at Leaf node
            - If know, follow to pointer child tree, else follow to parent
        - Insert of Entity E
            - Forward to the first node know E
        - Name resolution
            - 迭代式
            - 递归式
            -
- Synchronization 同步
    - 在单CPU系统中使用信号量(依赖shared memory)来解决
    - hard to determine order of event
    - 时钟同步：基于实际时间的同步

    - Lamport: Clock synchronization need not be absolute
    - all process agree on the order of events
    - UTC
        - Cs-133
        - average of 50 clocks around world
        - broadcast by satellite
    - One time receiver, other stay synchronized with receiver
        - changes gradually by add or rm seconds per interrupt
- Logical Clock
    - Happened-Before
        - a and b in same process, a -> b
        - a send msg to b, a -> b
        - a -> b, b -> c => a -> c
    - Implementation
        - 每个进程维护本地计数器，进程中每发生一个事件，计数器加一
        - 发送消息时，添加发送者计数器的值作为时间戳ts
        - 收到消息时，将本地计数器调整为max{ts, C} + 1
    - example: totally order multicast
        - each process has a queue
        - 当queue中存在来自其他所有进程且timestamp更大的消息时，队首消息被提交
- Vector Clock
    - a happened-before b => ts(a) < ts(b)
    - ts(a) < ts(b) => a happened before b ?
    - Pi add VC[i] when send msg, Pj update VC when recv msg
    - Pj 推迟消息投递到应用层直到
- Mutual exclusion 互斥访问
    - 集中式
        - Request + Grant + Release
        - 优点：公平，没有饥饿，实现简单
        - 缺点：协调者单点失效，性能bottleneck;
    - 分布式
        - queue by Lamport Vector Clock
        - implementation
            - sender: msg(CriticalRegion + ProcessNumber + TimeStamp)
            - receiver

          - 不在冲突域，也不想访问，reponse OK
          - 在冲突域，msg入队queue
          - 不在冲突域，想访问，如果msg的TimeStamp更小，response OK，else msg入队queue
      - 缺点
        - 单点失效 => 多点失效，一旦某个节点down，之后的任何进入冲突域请求再也无法通过（拒绝请求也发送恢复消息）
        - 每个节点需要自己维护group membership
        - 更复杂，代价更高，less robust
    - Token ring
    - Election Bully: process has weight
      - 向所有其他进程发送Election
      - 如果收到来自weight较低进程的Election，回复Take-Over
      - 如果进程没有收到Take-Over，赢得选举，发送Victory消息给所有其他进程
    - Election in Ring: process has weight
      - 发起者发送Election消息给后继
        - 如果后继crash，跳过
        - 如果没有crash，则后继将自己的编号加入Election消息中
      - Election回到发起者时，选择weight最高的编号为Leader，发送Coordinator消息绕环通知
- 复制与一致性
  - Replication 复制
    - 优点：避免单点失效，Scalability
    - 缺点：对Client是否透明，更新的代价高，不一致性问题
  - Consistency 一致性
    - Strict Consistency 严格一致性
      - Read操作返回最近一次的Write操作结果
      - 假设存在全局的时间order
    - Linearizability Consistency 线性一致性
      - 每个操作的生效带有一个全局的时间戳 (不一定为物理时间)，操作的顺序和时间戳一致
      - Read和Write操作按相同的顺序被执行
    - Sequential Consistency 顺序一致性
      - 与线性一致性类似，但没有全局的时间戳顺序
      - 从单个进程的角度来看，其指令的执行顺序与program order一致
      - 从多个进程的角度来看，指令的执行顺序一致
      - 各个线程类比成一根有弹性的绳子

- Causal Consistency 因果一致性
  - 当Read操作紧跟着一个Write操作时，这两个操作存在因果关系。读操作也与之前对相同变量的写操作存在因果
  - 所有的进程看到的因果关系一致
  - (a) 不满足，(b) 满足

- FIFO Consistency
  - 同一个进程的Write操作顺序，其他进程看起来是一致的
  - 不同进程的Write操作顺序，其他进程看起来可以是不一致的

- Models with Synchronization OP
  - Weak Consistency
    - 两种变量：同步变量 和 其他共享变量
    - 所有进程完成同步操作后，共享数据一致；否则共享数据可以为任意值
    - 对同步变量的访问满足 Sequential Consistent
    - 所有进程完成对同步变量的同步Write之后，在能对同步变量进行操作
    - 所有进程完成对同步变量的操作之后，才能对其他同步变量进行操作
  - Release Consistency
    - 离开临界区时，共享数据一致
    - 所有共享变量共用一个锁
    - 在获取一个锁操作之后，才能进行共享变量的读和写操作
    - 在释放一个锁操作之前，所有进程之前的读和写操作必须已经完成
  - Entry Consistency
    - 进入临界区时，共享数据一致
    - 每个共享变量都有一个锁
    - 在获取锁之前，所有与锁相关的操作必须执行完毕
- Client-Center Consistency
  - 如果有足够的时间，所有的副本将逐渐成为一致的
  - Monotonic Read 单调读
    - 如果读出了值A，那么之后读到的值只能是A或比A新的值
  - Monotonic Write 单调写
    - 多个副本间的写操作必须顺序进行，不能交叉
  - Read your Write
    - 某个进程对数据的写操作，总是可以被后续操作所看见
  - Write follow Read

            - 某个进程对数据的写操作，总是基于之前的读操作的值或更新的值
    - 副本间传播的信息
        - 更新的通知
        - 数据
        - 更新操作
    - Replicate by Pull or Push
    - Replication Protocols
        - Primary Backup Protocol 主备份协议：每个数据都有一个负责的主服务器
            - Remote-Write Protocol
                - Write(X) Operation Forward to Server who contains X
                - 先进行Servers之间的同步，再将Write结果返回给Client
            - Local-Write Protocol
                - Write(X) Operation Forward from Server who contains X to Server who process Operation
                - 可以先将Write结果返回给Client，再进行Servers之间的同步
        - Active Replication 主动复制
            - 操作被发送到所有副本，需要保证顺序的multicast
            - 问题：A调用B，那么对A的复制会使得所有副本都调用一次B
            - Quorum-Based Protocol
                - 所有读操作，都必须得到Nr个副本的同意；所有写操作，都必须得到Nw个副本的同意
                - Nr + Nw > N
                - Nw > N/2 避免写写冲突
                - ROWA：Read One Write All (Nr=1, Nw=N)

- Fault Tolerance 容错
    - Dependability 可信性
        - Availability 可用性
            - 多少的时间比例是可以被使用的
        - Reliability 可靠性
            - 在给定的时间内有多少的概率不会出错
        - Safety 安全性
            - 偶然的故障不会导致灾难性的后果
        - Maintainability 可维护性
            - 故障修复的容易程度

- Fault =(cause)=> Error =(result in)=> Failure
  - Failure 失败：系统保证的承诺
  - Error 错误：不正确的状态
  - Fault 故障：造成错误的原因
- Improve Dependability
  - Redundancy 冗余
    - Information Redundancy 校验位
    - Time Redundancy 重试
    - Physical Redundancy
      - Hardware
      - Software
    - Process Redundancy
      - Flat Group
        - Replicated-Write
      - Hierarchical: Master-Workers
        - Primary Copy
        -
  - K Fault Tolerant
    - K个组件故障
      - Fail-Slient faults: K+1个副本，总共2K+1个
      - Byzantine faults: 2K+1个副本，总共3K+1个
  - Communication Fault
    - Never reach Agreement
    - msg的发送者，不知道这条消息有没有被收到
  - Byzantine Fault
- Atomic Multicast 原子多播
  - 要么被发送给所有进程，要么就一个也不发送，且保证顺序相同
- Recovery 恢复
  - 从失败恢复到正确状态
  - Backward Recovery
  - Forward Recovery
  - Checkpoint 检查点
    - 定期持久化自身状态到磁盘上

            - Independent Checkpoint
                - 各个进程独立生成Checkpoint
            - Two-phrase Blocking Protocol
                - Checkpoint Request => Do Checkpoint => Checkpoint Done => ACK
            - Distributed Snapshot
- 云计算
    - 一种计算能力，提供了计算资源的抽象，通过网络，按需使用
    - 特点
        - 高可靠性
        - 通用性
        - 按需购买
        - 安全
        - 方便
    - 网格计算：利用互联网把地理上广泛分布的各种资源组成一个逻辑整体，像一台计算机一样为用户提供信息和服务
    - IaaS (Infrastructure as a Service):
    - PaaS (Platform as a Service): Spark, Hadoop, Google App Engine
    - SaaS (Software as a Service):
    - 虚拟化：位于下层的软件模块，提供物理和软件的接口，使得上层软件可以直接运行在其之上
        - 封装与隔离
        - 多实例
        - 硬件无关：整合异构硬件资源，虚拟机迁移
        - 特权功能：入侵检测和病毒检测
        - 动态调整资源：细粒度的可拓展性
    - 全虚拟化：VMM on Ring0，Guest OS on Ring1，APP on Ring3
    - 半虚拟化：修改客户机OS，将不可虚拟化的指令改为对HyperCall的调用
    - 硬件辅助虚拟化：VMM on Ring -1， Guest on Ring0
    - 云平台
        - 用户：按需使用各种类型的服务
        - 服务商
            - 提供各种类型的服务
            - 资源的合理配置
- 大规模阅读器的RFID系统，如何在分布式的环境下实现有效的数据感知，避免阅读器-标签冲突，阅读器-阅读器冲突
- 基于目标定位的微动作感知识别机制

- 按键操作检测
- 按键操作定位
- 低时延文本输出
    - 动态调整图片大小
    - 聚焦图片中的目标识别区域
    - 多线程
    - 剔除图片读写操作