并行与分布式计算 - June 22'th, 2018

Process Model Concepts

https://github.com/rh01

End-to-End Arguments in System Design[1]

- The communication network version of the end-to-end argument:
  - The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)
- Conclusions: It is rational for moving a function upward in a layered system closer to the application that uses the function, low-level mechanisms are justified only as performance enhancements.

Applications of End-to-End Arguments[2]

- The application of Occam's Razor
- Real-time video streaming over the Internet
  - Constructing it directly on the top of IP multicast contravenes the end-to-end principle
  - Adopt the end-system approaches (building an application-level multicast over the overlay) where control and intelligence reside at the edges of the network and not in the network itself.
- Google API: API designs should be as small as possible and no smaller
  - e.g. APIs should be easy to use and hard to misuse. It should be easy to do simple things, possible to do complex things, and impossible or at least difficult, to do wrong things

CAP Examples

- Amazon Dynamo
  - Dynamo: Amazon's Highly Available Key-value Store, 2007
  - C: eventual consistency
    - Gossip-based membership protocol
  - A: data replication with (N, R, W)
    - Quorum-based: R+W>N
  - P: consistent hashing
- Facebook/Apache Cassandra
  - Cassandra: a decentralized structured storage system, 2010
  - C: eventual consistency
    - Gossiping

- A: data replication with (N, R, W)
  - Quorum-based: R+W>N
- P: DHT

分布式计算的模型-1-POSET

- Formally, a distributed computation can be modeled as the partially ordered set (poset) $\hat{H}$ =(H, $\Rightarrow$), where, H is the set of all event, $\Rightarrow$ is the binary relation denoting causal precedence between events defined as follows:

$$e_i^a \rightarrow e_j^a \equiv \begin{cases} (i=j) \wedge (b=a+1) \\ (e_i^a = send(m)) \wedge (e_j^b = receive(m)) \\ \exists e_k^c : (e_i^a \rightarrow e_k^c) \wedge (e_k^c \rightarrow e_j^b) \end{cases}$$

- Let

$$h_i = e_i^0, e_i^1 \ldots$$

be the local history of process P_i.

分布式计算的模型-2-DAG

- Let ai be the local state of process P_i immediately after having executed event e_i^a, and let S be the set of all local states
- With respect to the set of local states, a distributed computation can be modeled as a DAG, denoted $\hat{S}$ = (S, $\prec$), where $\prec$ is the binary relation denoting immediate causal precedence between local states as follows：

$$\sigma_i^a \prec \sigma_j^b \equiv \begin{cases} (i=j) \wedge (b=a+1) \\ (a_i^{a+1} = send(m)) \wedge (e_j^b = receive(m)) \end{cases}$$

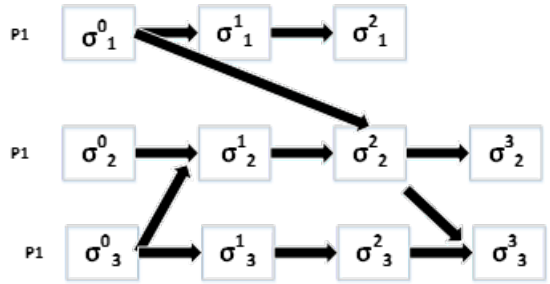

图 1：DAG

概论

- 物理模型：描述系统的硬件组成
- 体系结构模型：用计算/通信任务来描述一个系统
- 交互模型：描述分布式系统的实体 (例如，进程) 是如何交互的，无全局时间和通信性能会带来怎样的影响
- 故障模型：定义了故障可能发生的方式
- 安全模型：对攻击进行定义和分类，考虑如何保护系统不被干扰或窃取数据

- 数据模型：描述数据的组织、访问方式会带来什么问题

物理模型

- 一组联网的可扩展的计算机节点
- 静止节点 ⇒ 移动节点
- 分立的节点 ⇒ 嵌入到其他物理实体内的节点
- 自治的节点 ⇒ 一组节点共同协作

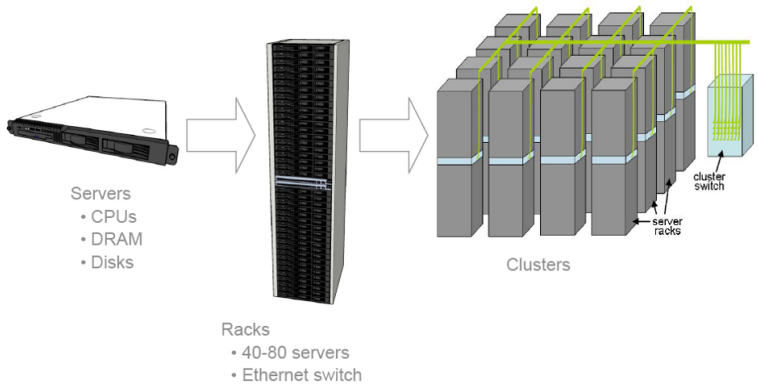

图 2：Google 的物理模型

- 在每个机架上安放 40 到 80 台商用 PC。每个机架有一台以太网交换机，提供机架之间和到机房之外的连接；
- 机架被组织成集群，一个集群通常由 30 个或更多的机架和两个高带宽交换机组成
- 集群被安置在 Google 遍布全世界的数据中心中

Organization of the Google physical infrastructure

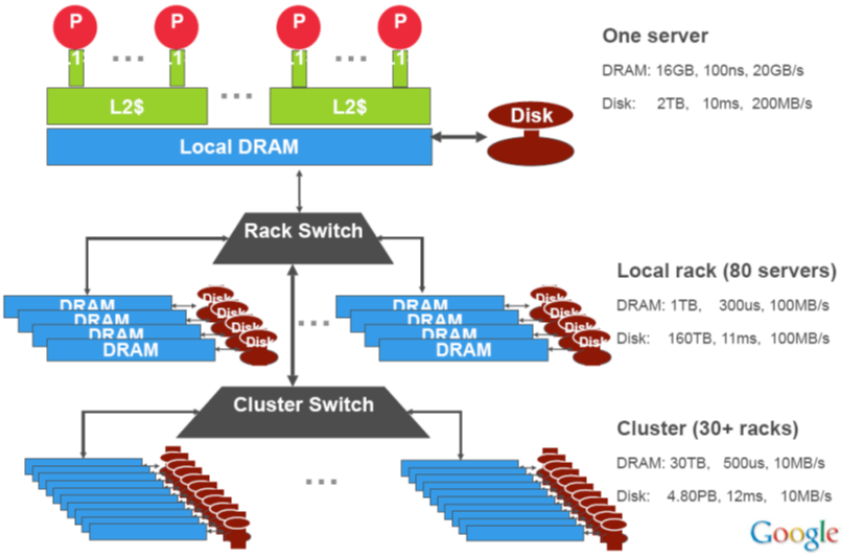

图 3：Organization of the Google physical infrastructure

Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA→Netherlands→CA	150,000,000 ns

表 1: Numbers Everyone Should Know

体系结构模型

- 体系结构：实体 (组件) 和实体 (组件) 之间的关系
- 在分布式系统中，什么是进行通信的实体？
  - 进程或线程；分布式对象；组件；Web 服务
- 它们如何通信，使用什么通信范型？
  - 进程间通信；远程调用 (请求应答协议，远程过程调用，远程方法调用)；间接通信 (组通信，发布/订阅，消息队列，元组空间，分布式共享内存)
- 它们在整个体系结构中扮演什么角色和责任？
  - 客户-服务器 (两层，三层，代理，多服务器)，P2P
- 它们怎样被映射到物理分布式基础设施上？
  - 放置需要考虑实体间的通信范型、给定机器的可靠性和它们当前的负载、不同机器之间的通信质量等等
- 关注：将服务映射到多个服务器；缓存；移动代码；移动代理

分布式对象、组件以及 Web 服务

- 分布式对象：
  - 基于对象方案固有的封装很适合分布式编程
  - 数据抽象的相关特性为对象的规约和其实现提供了清晰的分离，它允许程序员只处理接口而无需关注诸如使用的编程语言和操作系统这样的实现细节。
  - 这种方式也有助于更具有动态性和可扩展性的解决方案，例如通过允许引入新的对象或者使用一个对象替代另一个兼容的对象
- 问题：

  - 隐式依赖
  - 编程复杂性
  - 缺少关注点分离支持
  - 无部署支持
- 组件：一个具有指定接口契约和仅有显式上下文依赖的组合单元
  - 一组提供的接口和一组所需的接口
  - 容器：组件的一个生存环境。支持容器模式和该模式蕴含的关注点分离的中间件被称为应用服务器
- 实现依赖显式化以及简化分布式应用的开发和部署
- Web 服务具有一个服务接口，该接口可提供操作来访问和更新其管理的数据资源
  - Web 服务不能创建远程对象的实例
  - 对象和组件经常在一个组织内部使用，web 服务通常被看成完整的有自主权利的服务，经常跨组织边界，用于实现业务的集成

体系结构模式 (patterns)

- 分层体系结构 (layering/layered systems)
- 层次化体系结构 (tiered architecture/leveling structure)
- Interceptor (From POSA2)
- Microkernel
- Reflection……

分层体系结构

- 系统分层是解决复杂软件系统设计的途径之一，使得系统具有高内聚性和低耦合性，避免系统具有不必要的依赖关系。
- 分层系统中，每层都有自己的组成元素，有自己的语法和规则，具有完整描述所需系统的能力。每一层通常可看成是一个虚拟机，利用下层的服务为它的上层提供服务。连接器则通过协议来定义层次之间的交互
- 应用
  - 分层通信协议
  - 分布式系统中服务的分层
- 优点
  - 支持渐进抽象层次的设计
  - 分层的可重用：只要到相邻层的接口相同，支持相同层的不同实现
  - 局部的依赖关系：一层只和上下层交互，改变一层的功能只影响上下两层
- 缺点
  - 难以建立抽象层次，不是所有的系统都可以简单构架为分层结构
  - 效率较低

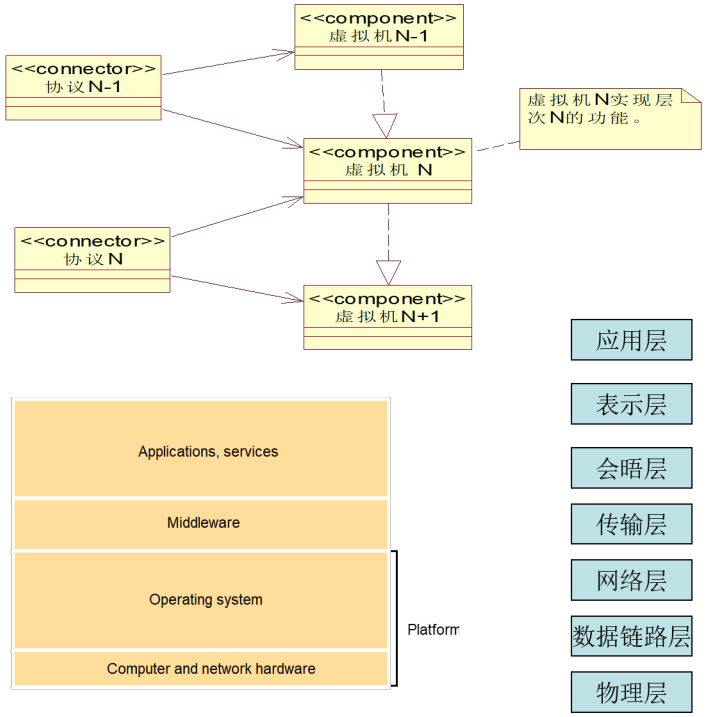

图 4：分层体系结构

层次化体系结构

- 关键是功能的划分和放置
- 两层结构：客户，服务器
- 三层结构：
  - 客户，应用服务器，数据库服务器

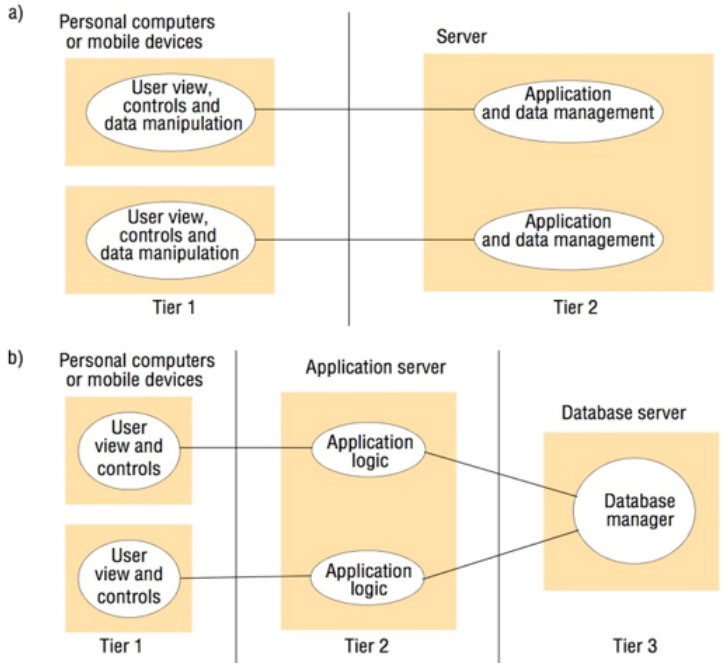

图 5：兩層和三層結構

- 客户/服务器两层结构的局限
  - 客户数超过 100，系统性能急剧下降；
  - 实现业务逻辑的时候通常使用特定厂商的数据库过程，这就限制了应用系统的灵活性和对 DBMS 的选择；
  - 目前的两层结构实现对于系统功能/负载在服务器之间的迁移的较少；
  - 适用于局域网，不足以扩展到大企业广域网或 Internet；
  - 对于软硬件的混成组合支持不足；
  - 难于进行客户管理，客户包含很多业务逻辑，难以安装维护，系统封装不好等

P2P 系统：特点

- 分散性：资源的所有权和控制权被分散到网络的每一个节点中
- 自治性：每一个 Peer 都是平等的参与者，扮演服务使用者和服务提供者两个角色
- 动态性：由于没有中央节点的集中控制，系统的伸缩性较强，能适应网络拓扑结构的变化，也能避免单点故障，提高系统的容错性能
- 高效性：服务使用者和服务提供者之间进行直接通信，可充分利用网络带宽从而减少网络的拥塞状况，使得资源的有效利用率大大提高

P2P 应用

- 资源共享主要是文件共享系统、文件分发系统（File Distribution System）
  - 典型系统有 Napster，Gnutella，FreeNet，Chord-based System，BitTorrent
- P2P 协作应用
  - 即时消息系统有 AOL AIM、Yahoo Messenger、MSN Messenger、Jabber 等
  - 在线游戏有星际争霸、Net-Z 和 DOOM
  - 共享企业应用有 Groove、Magi
  - 文件搜索有 YouSearch, OpenCola 等
  - pub/sub 系统有 Scribe、Herms 等
- 科学计算系统：Seti@Home – Search for Extraterrestrial Intelligence

P2P 系统的若干关键问题

- 通信和定位
  - P2P 网络的拓扑结构和 Peer 节点的功能角色划分
  - 资源和服务如何标识
  - 进行资源查找时如何进行 Peer 定位
  - P2P 网络中 Peer 节点动态变化的处理
  - 如何穿越 NAT (Network Address Translation) 和防火墙进行 Peer 节点之间的直接通信
- 安全问题：匿名性
- 服务质量：请求响应时间、公平性

P2P 系统的拓扑结构

- 纯 P2P 网络，每个 Peer 都具有同等的责任和地位，不存在中间节点的协调
  - FreeNet、Gnutella 都属于纯 P2P 网络
- 混杂的 P2P 网络，存在着充当服务器角色的 Peer 节点或提供特殊功能的 Super-Peer 节点
  - Napster, Groove, Magi 等系统均是典型的混杂型 P2P 系统

Peer 节点的角色划分

- 简单类型 Peer 节点仅仅是一个简单的终端用户，它可以请求获得服务并为网络中的其它 Peer 提供服务
- Super-peer 节点除了具有和简单 Peer 节点类似的功能外，还提供某些特殊功能
  - 路由器 Peer 节点作为一个桥梁，使得被防火墙或 NAT 隔离的 Peer 可以相互通信
  - 会聚点 Peer 节点为简单节点提供查询定位信息的功能
- 服务器型的 Peer 节点主要提供类似于客户/服务器模型下的服务器功能
  - 在 Napster 中，有若干个简单 Peer 节点相互提供文件下载的服务，还有一个目录服务器节点提供文件条目的注册管理
  - Groove 和 Magi 系统中也均存在这样的服务器型节点

Peer 的定位方式

- 直接定位 Peer 即利用广播或多播的形式发出查询请求，符合查询要求的 Peer 节点进行应答，然后建立直接的通信连接
- 间接方式
  - 服务器模型：基于混杂型的 P2P 拓扑结构
  - 泛洪模型：基于纯 P2P 拓扑结构
    - 可以用 TTL(Time to Live) 或 HTL(Hops to Live) 进行转发控制
  - 路由模型：基于纯 P2P 网络结构
    - 非结构化路由模型
    - 结构化路由模型

防火墙和 NAT 的穿越

- 需要使用在一般情况下可以通过防火墙的协议，如基于请求/应答方式的 HTTP 协议
- Peer 之间进行通信时，必须由内部网络的机器首先发起连接请求，如果通信双方均处于防火墙之后，则需要由防火墙外的转发节点进行消息转发

安全问题之匿名性

- 发布者匿名，请求者匿名，服务提供者匿名
- 提供匿名性的基本方法
- 组播：采用 IP 组播方法进行资源发送可以达到请求者匿名的目的
- 地址欺骗：使用面向无连接的协议如 UDP

- 身份欺骗：网络中的文件可能拥有多个备份，所以应答者往往并不是文件的提供者
- 隐藏路径：Peer 之间并不是进行直接的消息通信，而是通过若干的中间节点
- 别名：每一个 Peer 在网络中均有一个别名，Peer 之间通过别名进行消息传递
- 强制存放：一个文件存放的位置是由文件本身确定的，发布者无法选择，从而实现发布者匿名，如基于 Chord 的系统

交互模型

- 无全局时间会给交互带来的影响：事件乱序
  - 事件排序：物理时间同步；逻辑时间 (向量时钟)
- 确定当前系统环境下，通信的延迟、所需带宽和抖动特性
- 确定系统对进程执行、消息传递、时钟漂移有无时间限制
  - 同步交互模型
    * 已知进程执行每一步的时间有一个上限和下限；通过通道传递的每个消息在一个已知的时间范围内 (min, max) 接收到；每个进程有一个本地时钟，它与实际时间的偏移率在一个已知的范围内
    * 进程可以达成协定。例如：前一支部队冲锋后，不超过 max-min+1 分钟，另一支部队保证发起冲锋 (Pepperland 协定)
  - 异步交互模型
    * 对下列元素没有限制：进程执行速度；消息传递延迟；时钟偏移率
    * 在异步系统中达成协定是有一定困难的：如果系统出现有故障，那么在有限步达成协议是不可能的。例如，联系两支部队的通信兵：被敌人抓住或者被敌人洗脑
- 了解系统有怎样的进程及数量、希望这些进程如何协调，从而决定通信范型
- 根据参与协同的进程之间的时序耦合程度和引用耦合程度，可以将两个进程之间的协调方式分成 4 类：direct, mailbox, meeting-oriented, generative communication
  - 时序耦合度的判定是根据通信的两个进程是否均已启动且正在运行
  - 引用耦合的判定是根据是否需要明确知道/了解通信的双方
  - 如果两个进程是时序上紧耦合和引用紧耦合的，那么进程之间的协调是直接协作方式：请求-应答，RPC，RMI
  - 如果两个进程是时序上松耦合和引用紧耦合的，那么进程之间的协调是邮箱协作方式：MOM
  - 如果两个进程是时序上紧耦合和引用松耦合的，那么进程之间的协调是会议型协作：Pub/Sub
  - 如果两个进程是时序上松耦合和引用松耦合的，那么进程之间的协调是生成通信型协作：一组独立进程利用（放入或检出）一组共享的持久的元组数据空间
- 根据进程交互双方的数量：两个，多个，全体
  - 应用层覆盖网 (overlay network) 上的交互方式：
    * 点-点通信 (unicast)
      · 特例：地理路由 geocast，选播 (anycast)
    * 多播 (multicast)
    * 广播 (broadcast)
      · 汇聚传输 (与广播对偶，convergecast，覆盖全体进程)

故障模型

Dependable System

- Diversification of concepts
  - high confidence software(高可信软件)
  - trustworthy computing(可信计算)
  - software dependability(软件可靠性)
- Dependability = reliability + availability + safety + maintainability + security
- High confidence = reliability + safety + security + survivability + performance
- Trustworthy = reliability + security + privacy + integrity

Concepts explained

- Reliability: a system can run continuously without failure
- Availability: a system is ready to be used immediately
- Metrics for reliability: MTBF = MTTF+ MTTR
- Metrics for availability: MTTF/(MTTF + MTTR) * 100%
  - MTBF: Mean Time between Failure; MTTF: Mean Time to failure; MTTR: Mean Time to Repair

Percent Availability	Downtime/Year	Categories
99.5	1.8 days	Conventional system
99.9	8.8 hours	Available system
99.99	52.6 mins	Highly available system
99.999	5.3 mins	Fault resilient system
99.9999	32 seconds	Fault tolerant system

图 6: Metrics

故障模型

- 名词定义：fault，error，failure
  - Fault：指系统的缺陷，有缺陷不一定会出故障
  - Error：指系统应该的状态与实际状态之间的差异
    - 当系统由于激活了 Fault，而进入一个意外的状态，那么出现 Error
  - Failure：指系统行为与其规约不一致
    - Error 未必引起 Failure：如果系统抛出的意外（Exception）被捕获，并被容错处理，那么系统的操作仍与规约一致
  - Fault(错误的原因) ⇒ Error(错误的系统状态) ⇒ Failure(错误运行后的结果)
- 故障分类
  - 瞬态性 (Transient)，间歇性 (Intermittent)，永久性 (Permanent)
- 故障分类：在进程和通信通道上发生故障

- 遗漏故障 (崩溃，故障-停止，通道遗漏故障，发送遗漏故障，接收遗漏故障)
- 随机故障
- 时序故障 (时钟故障，进程性能故障，通道性能故障)

故障描述

故障类型	影响	描述
故障-停止	进程	进程停止并一直停止。其他进程可检测这个状态
崩溃	进程	进程停止并一直停止。其他进程可能不能检测到这个状态
崩溃	通道	通道断链并一直断链。可能不能检测到这个状态
通道遗漏	通道	插入外发消息缓冲区的消息不能到达另一端的接收消息缓冲区
发送遗漏	进程	进程完成了发送操作，但消息没有放到它的外发消息缓冲区
接收遗漏	进程	一个消息已放在进程的接收消息缓冲区，但那个进程没有接收到它
随机	进程	进程/通道显示出随机行为：它可能在随机时间里发 (拜占庭式) 和送/传递随机的消息，会有遗漏发生；一个进程可能
	通道	停止或者采取不正确的步骤
时钟	进程	进程的本地时钟超过了与实际时间的偏移率的范围
性能	进程	进程超过了两个进程步之间的间隔的范围
性能	通道	消息传递花了比规定的范围更长的时间

表 2: 故障描述

故障的处理：基本手段

- 故障检测技术
  - 被动式检测是在程序中的若干位置设置检测点，等待错误征兆的出现
    - 需要提供与程序执行模块相分离的自动错误检测模块
  - 主动式检测是对程序状态主动地进行检查；
    - 利用心跳判断分布式进程失效：只是指出进程 p 可能失效，并非精确地指出 p 真正失效了：不可靠的故障检测器
    - eventually weak failure detector (eventually weakly complete, eventually weakly accurate)：这样的故障检测器在通信是可靠的、崩溃进程不超过一半的时候，能在异步系统中达成共识
- 容错 (故障屏蔽)：利用信息冗余、时间冗余、物理冗余
  - 通过信息冗余，例如，校验和，将通信的随机故障转化为遗漏故障
  - 通过时间冗余，例如，消息重传，事务
  - 通过物理冗余，例如，用多个进程提供一个服务，集群
- 故障恢复机制
  - 故障恢复包括 4 个步骤：
    - 故障诊断 (fault diagnosis)
    - 故障隔离 (fault isolation)

    * 系统重配置 (system reconfiguration)
    * 系统重启 (system re-initialization)
  - 故障恢复手段：
    * 进程迁移
    * 进程替换

安全模型

- 分布式系统的安全：使进程和用于进程交互的通道获得安全，并保护所封装的对象不被未经授权的访问
- 识别安全威胁：包括对进程的威胁、对通信通道的威胁
  * 对进程的威胁：对消息源缺乏可靠的知识是对服务器和客户正确工作的一个威胁
  * 对通信通道的威胁：一个敌人在网络和网关上能拷贝、改变或插入消息；另一种形式的攻击是试图保存消息的拷贝并在以后重放，即反复重用同一消息；敌人通过超量地、无意义地调用服务或在网络上进行消息传递，干扰授权用户的活动，导致物理资源 (网络带宽，服务器处理能力) 的过载—服务拒绝

数据模型

- 数据模型刻画数据的组织和访问方式
  * 数据查找
    · 域名查找/名字服务
    · 搜索引擎
    · P2P 数据的定位
  * 数据的复制和一致性
  * 静态负载分配、动态负载分配
  * 数据的并发控制策略
    · 锁，时间戳，乐观并发控制及其分布式策略
    · 分布式死锁的预防、避免和检测
  * 无用单元收集