分布式计算

7 个题

论文 2 个 FLP、Paxos Caft

课上 1 个

简答 5 个

P2P 实践 bitTorent

云计算 2 个 概念、技术—虚拟化→虚拟机

概述 2 个 概念→进程通信

灵活答题、不要死记硬背

卷面整洁 字要写直

概述

分布式计算：如何把一个需要巨大的计算能力才能解决的问题分解成许多小的部分，然后把这些部分分配给许多计算机进行处理，最后把这些计算结果综合起来得到最终的结果。

交互应用 + 支撑技术（网络、CPU、存储） 驱动 分布式计算快速发展

优缺点

优点：

低廉的计算机价格和网络访问的可用性：大量的互联计算机为分布

式计算创建了一个理想环境

资源共享：有效的汇集资源

可伸缩性：对资源需求的增加可以通过快速提供额外资源来有效解决，如增加计算机

容错性：故障情形下依旧提供资源访问

缺点：

安全性低：非集中式管理、非授权用户多

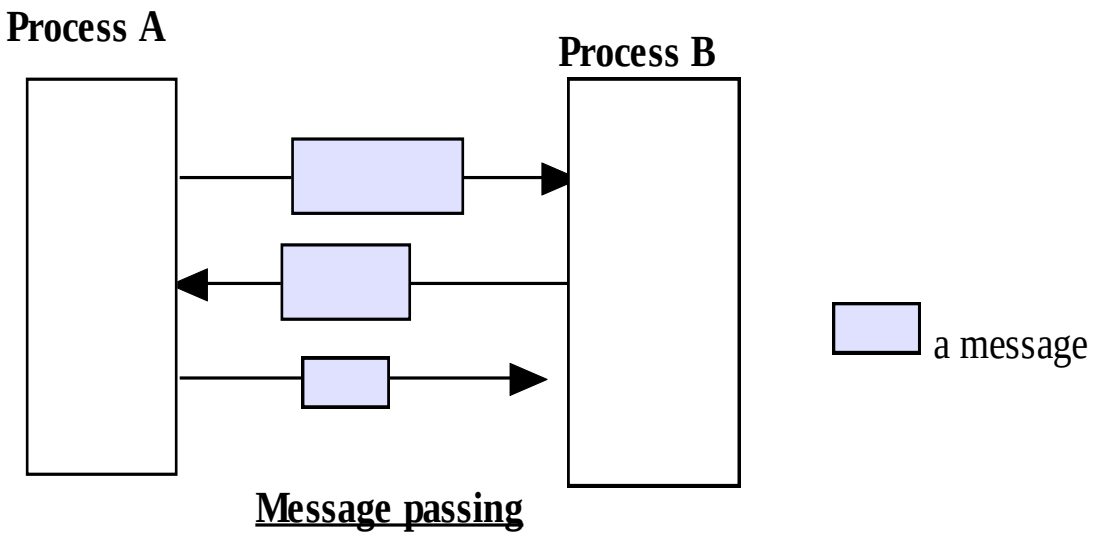
多点故障：计算机和链路故障引发分布式系统故障

分布式计算特征

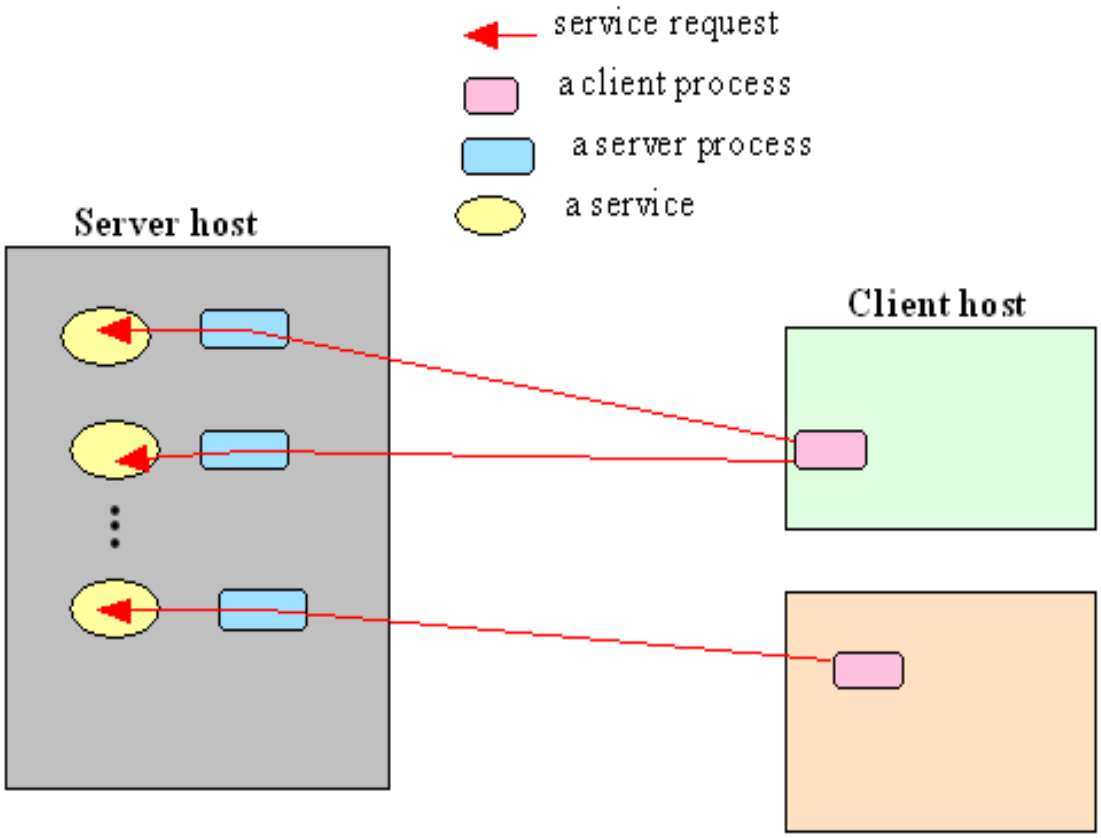
- 可靠性：指一个分布式系统在它的某一个或多个硬件的软件组件造成故障时，仍能提供服务的能力。
- 可扩展性：指一个系统为了支持持续增长的任务数量可以不断扩展的能力。
- 可用性：指一个系统尽可能地限制系统因故障而暂停的能力。
- 高效性：指一个分布式系统通过分散的计算资源来实现任务执行的高效率。

范型

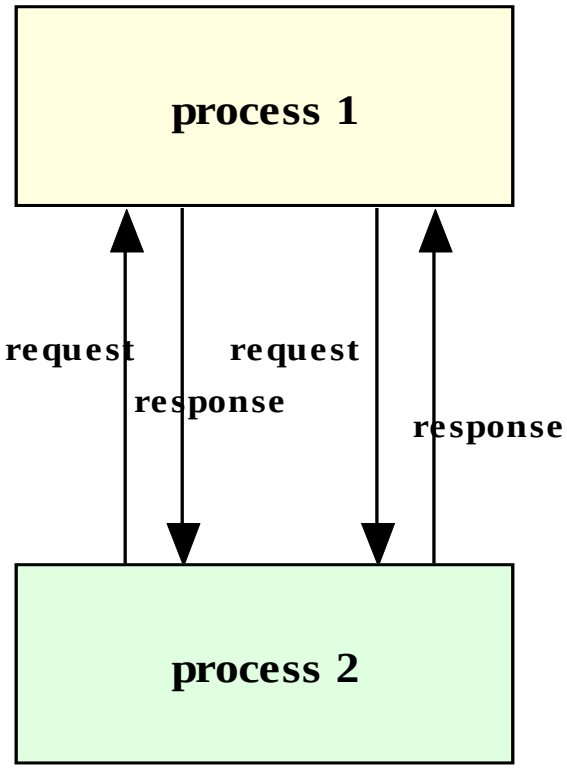
1.消息传递



2. 客户-服务器范型



3. Peer-to-Peer 范型



4. 消息系统

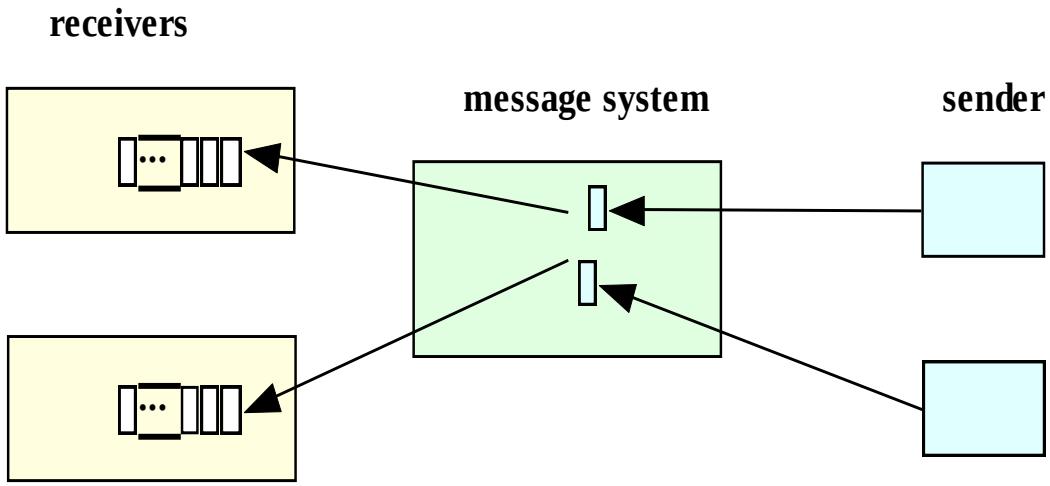
◆ 点对点消息模型（Point-to-point message model）

消息系统将来自发送者的一条消息转发到接收者的消息队列中。这种中间件模型提供了消息暂存的功能，从而可以将消息的发送和接受分离。点对点消息模型为实现异步消息操作提供了额外的一层抽象，可以借助于线程或者子进程技术。

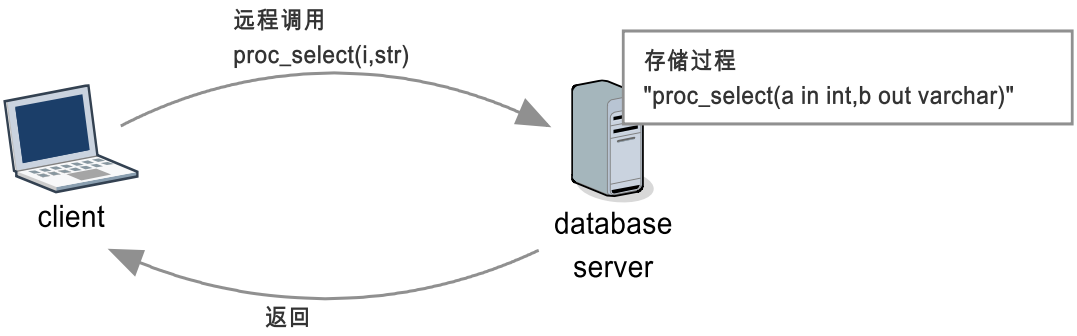
◆ 发布订阅消息模型（Public/Subscribe message model）

每条消息都与某一主体或事件相关。对某个事件感兴趣的应用程序可以订阅与该事件相关的消息。当订阅者等待的时间发生时，触发该事件的进程将发布一条消息来宣布该事件或主题。中间件消息系

统将这条消息分发给该消息的所有订阅者。发布/订阅消息模型提供了一种用于组播或组通信的强大抽象机制。



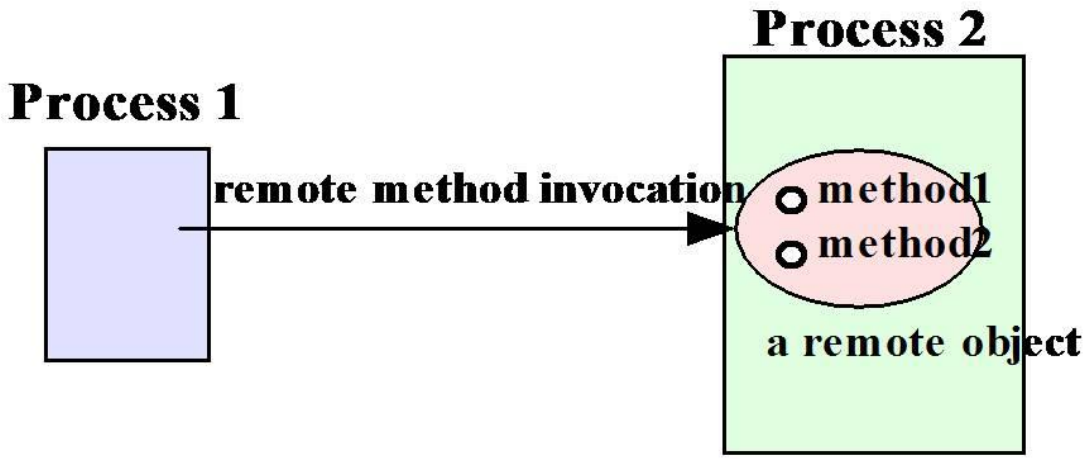
5. 远程过程调用



报文：传参数

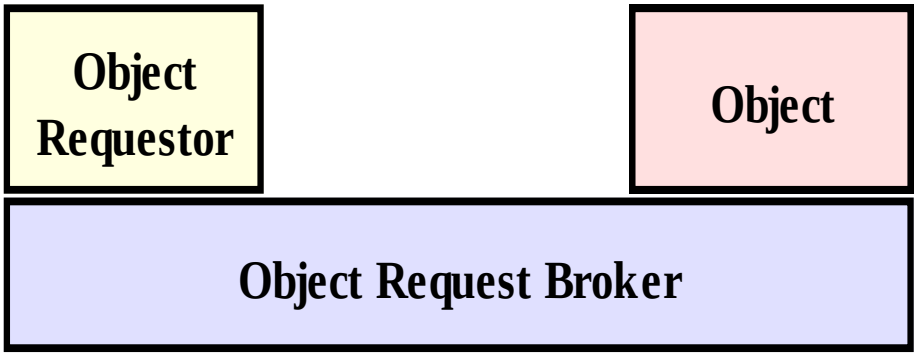
代理：根据报文运行进程

6. 分布式对象

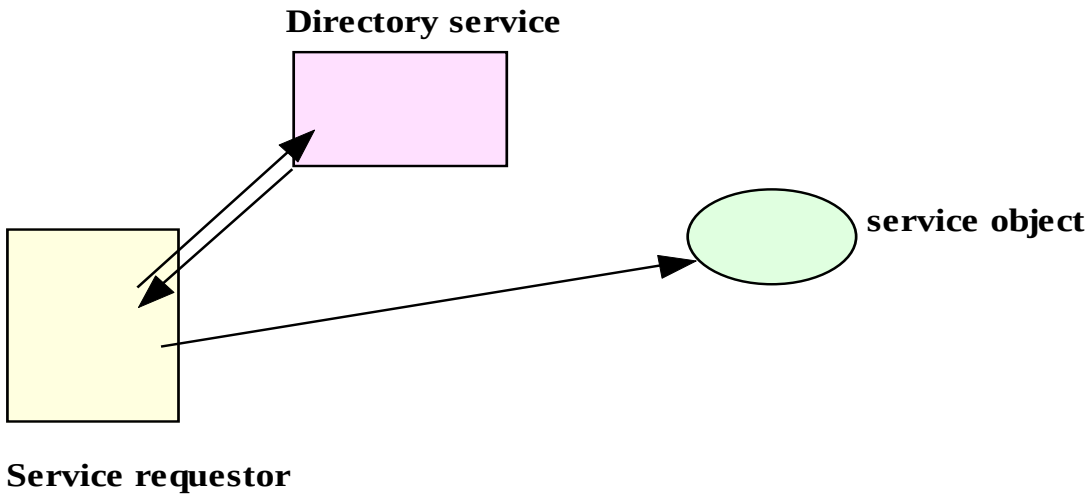


远程方法调用（**Remote Method Invocation，RMI**），进程调用对象方法

7. 对象请求代理



8.网络服务



由服务请求者、服务提供者（对象）和目录服务三者组成。

本质是 RMI 扩展

9.云服务范型 之 IaaS

云服务范型 之 PaaS

云服务范型 之 SaaS

进程间通信

分布式计算的核心：**Inter-Process Communication，IPC，**即在互相独立的进程间通信及共同协作以完成某项任务的能力

- **单播：一个进程与另一个进程进行 IPC（unicast）**
- **组播：一个进程与另外一组进程进行 IPC（multicast）**

同步 send 和 同步 receive（后 先）

异步 send 和 同步 receive（后 先）

同步 send 和 异步 receive（先 后）

异步 send 和 异步 receive

死锁和超时

用线程实现异步操作

事件状态图

IPC 范型

Peer to Peer Computing P2P

P2P 是一种分布式网络，打破传统的 C/S 模式，在网络中的每个结点的地位都是对等的。每个结点 peer 既充当服务器，为其他结点提供服务，同时也享用其他结点提供的服务

拓扑架构

Napster 中心化拓扑

Gnutella 全分布式 非结构化拓扑

Chord 完全分布式 结构化拓扑

Kazaa 半分布式 结构化拓扑

BitTorrent

实体节点:

- 索引服务器 Tracker

- 下载节点 Leecher：C、S

当 leecher 下载包含多个文件时→服务器

- 种子节点 Seed：只充当服务器

功能协议:

Tracker 协议

服务器-seed 开始通信、心跳包

Peer 协议

建立连接

交换位图

片段选择

请求与传输

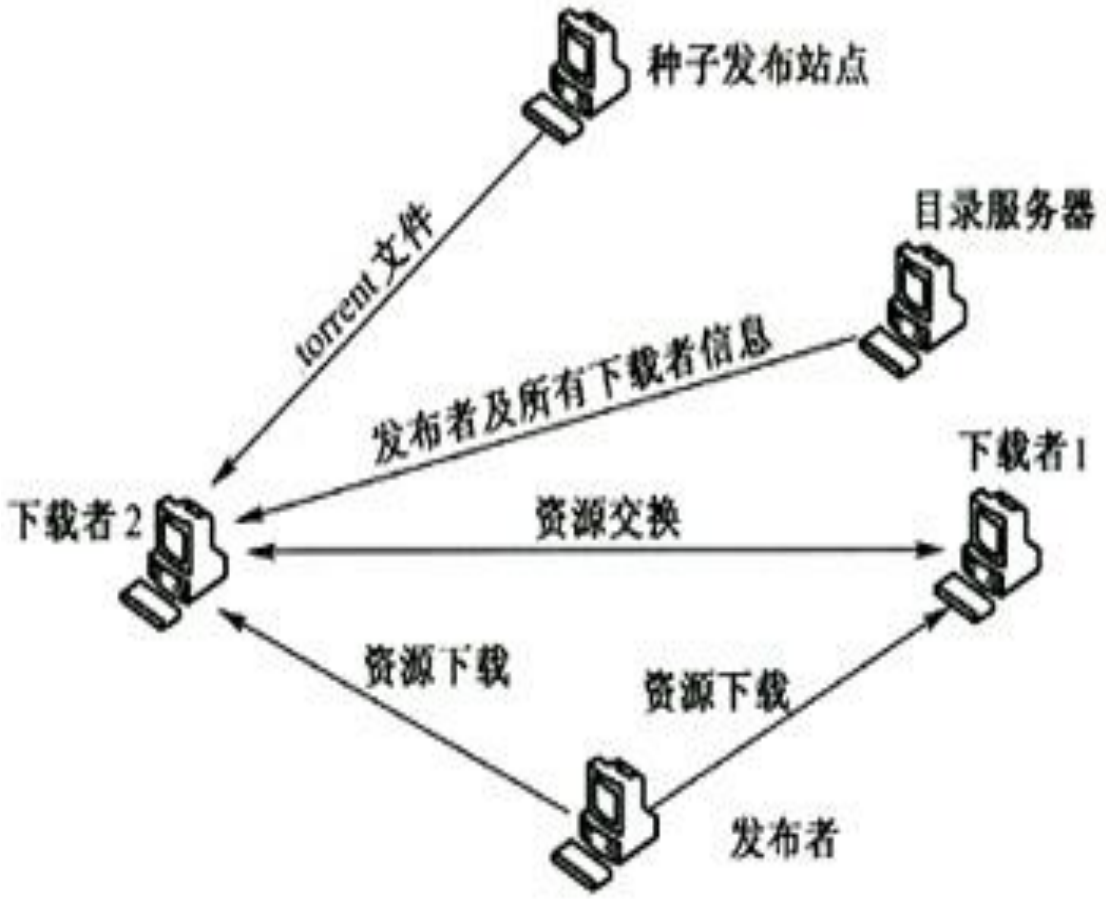
片段校验、通知周边节点、提高资源利用率

关键算法:

Choking 算法

上传带宽、新节点表现期

片段选择算法



云计算

云计算包括互联网上各种服务形式的应用，以及这些服务所依托数据中心的软硬件设施。

云计算**核心思想**是根据用户需求，将大量用网络连接的计算、存储资源集中进行统一管理和调度，构成一个资源池，向用户提供 IT 基础设施、数据和应用的服务，这个提供资源的网络就称为“云”。

本质

云计算是一种面向效用的和因特网为中心的**按需交付 IT 服务**的方式。这些服务覆盖了所有的计算栈：从打包的硬件基础设施作为一

系列虚拟机，到软件服务例如开发平台和分布式应用。通过在供应商和顾客之间服务谈判，它能动态提供和呈现作为一个或更多以建立的服务级协议为基础的统一的计算资源

需求

- 接入能力

可以从任何地点、任何设备接入服务和数据

- 共享能力

数据的建立和存储共享 容易方便

- 自由

不希望受数据的影响

- 简单

容易学会, 容易使用

- 安全

相信数据不会丢失或不会被不允许的人看到

判断是不是云计算的三条标准

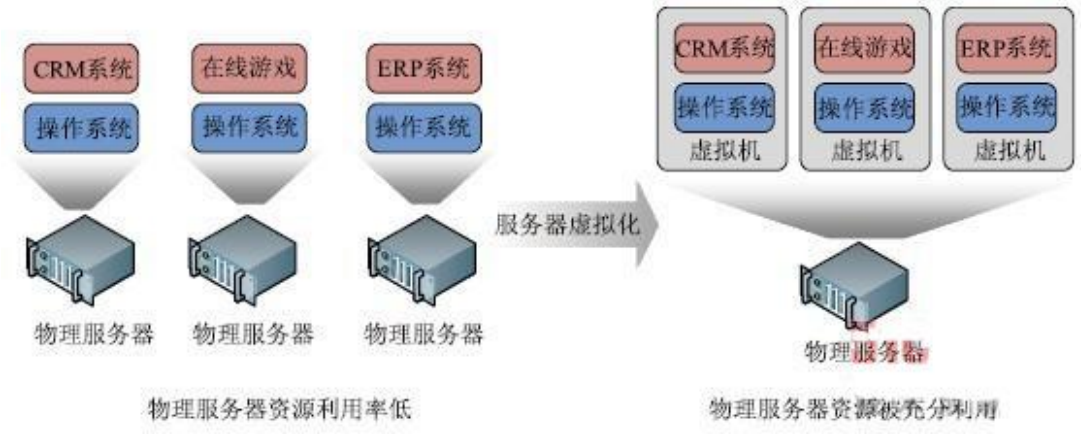
1. 用户使用的资源不在客户端而在网络中。
2. 服务能力具有优于分钟级的可伸缩性。
3. 五倍以上的性价比提升。

虚拟化

服务器虚拟化

服务器虚拟化将系统虚拟化技术应用于服务器上，将一个服务器虚

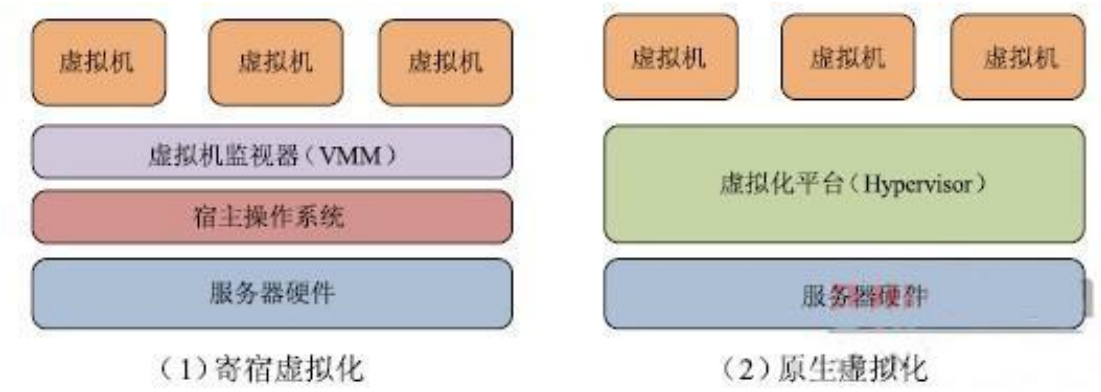
拟成若干个服务器使用



两种类型：

寄宿虚拟化

原生虚拟化



	寄宿虚拟化	**原生虚拟化**
是否依赖于宿主操作系统	完全	不
性能	低	高
实现的难易程度	易	难

关键特性：

多实例、隔离性、封装性、高性能

CPU 虚拟化

把物理 CPU 抽象成虚拟 CPU，任何时刻一个物理 CPU 只能运行一个虚拟 CPU 的指令

基于分时

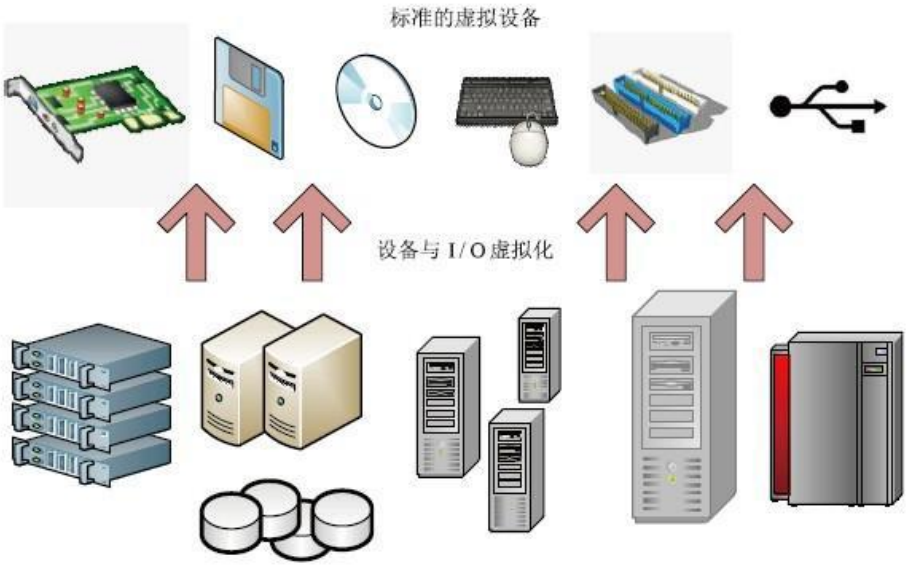
全虚拟化	半虚拟化
• 采用二进制代码动态翻译技术（敏感指令前插入陷入指令） • Microsoft Virtual PC , VMware WorkStation	• 修改客户操作系统来解决虚拟机执行特权指令的问题 • Citrix Xen， VMware ESX Server

内存虚拟化

把物理机的真实物理内存统一管理，包装成多个虚拟机的物理内存给若干虚拟机使用

设备和 I/O 虚拟化

通过软件的方式实现



不同种类的服务器和存储设备

实时迁移技术

原因：宿主故障、虚拟机过多

- 实时迁移技术是在虚拟机运行过程中，将整个虚拟机的运行状态

完整、快速地从原来的宿主机硬件平台迁移到新的宿主机硬件平台上

- 整个过程是平滑的，用户不会察觉
- 需要虚拟机监视器的协助

技术优势

- 降低运营成本
- 提高应用兼容性
- 加速应用部署
- 提高服务可用性
- 提升资源利用率
- 动态调度资源
- 降低能源消耗

云计算的特点

☞**超大规模**

☞**虚拟化**

☞**高可靠性**

☞**高可伸缩性**

☞**按需服务**

☞**极其廉价**

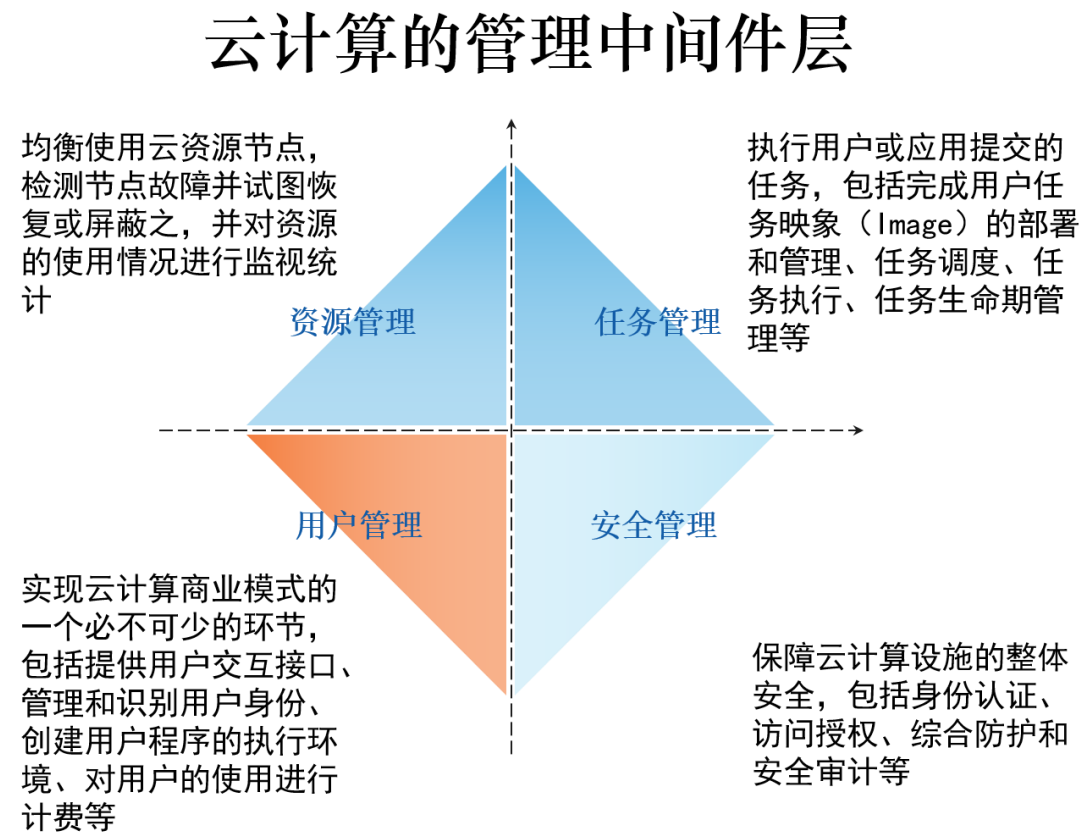
云计算的服务模型

SaaS 软件即服务

PaaS 平台即服务

IaaS 基础设施即服务

中间件



体系结构

云计算关键技术

资源调度

多租户技术

海量数据处理

大规模消息通信

大规模分布式存储

许可证管理与计费

FLP

衡量共识算法正确性的标准

- Termination（终止性）

非失败进程最终可以做出选择

- Agreement（一致性）

所有的进程必须做出相同的决议

- Validity（合法性）

进程的决议值，必须是其他进程提交的请求值

满足以上三个条件的算法，就可以称之为"共识算法"。

FLP 不可能性就是在异步模型下证明的

异步模型的主要特点是：

1. 进程之间的通信基于消息（message）。消息的传输是可靠的，即 消息可能延迟，可能乱序，但最终都会到达。

2. 没有时钟（clock）可用，因此进程无法做出消息超时的判断。

3. 进程的存活状态是无法检测的，即外部不能区分进程运行缓慢或是终止的情况；

假设：

1.不考虑拜占庭类型的错误（叛徒发送前后不一致的提议，被叫做拜占庭错误）

2.最多只有一个进程失败

Lemma1 交换性

把所有的进程的集合 U 分成两个不想交的集合 P1，P2，有两个 执行序列：σ1，σ1 分别作用在 P1，P2 的集合上，结果不变。

定理一：在有一个进程出错的情况下，不存在一个完全正确的一 致性协议。

Lemma2 初始 Configuration 的不确定性

引理

1. 如果两个操作序列 操作的是不同的进程集合，那先执行哪个序列结果都一样这个。（前提是 C 经过某个 e，结果是确定的）

<连通性>

2. P 具有一个不确定的初始 configuration

<初始不确定性>

3. 从一个不确定的 Configuration 执行一些操作后仍可能得到一个不确定的

Configuration

<不可终止性>

FLP 结论

在异步通信场景中，即使只有一个进程失败，也没有任何算法能保证非失败进程达到一致性。

Paxos

Paxos 是基于消息传递且具有高度容错特性的一致性算法

前提假设是不存在拜占庭将军问题，即：信道是安全的（信道可靠），消息都是完整的，没有被篡改。

在可能发生异常的分布式系统中，在集群内部对某个数据的值（value）达成一致

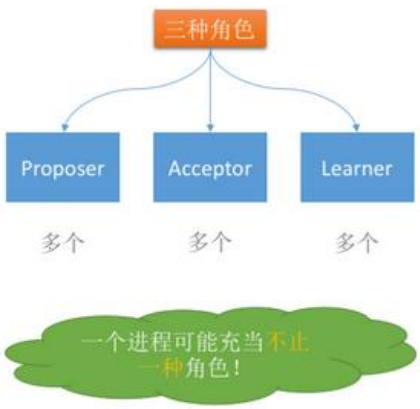
三种角色

Proposer 负责提出提案(Proposal)

Acceptor 负责对提案做出裁决(是否 accept)

Learner 负责学习提案结果

Proposal 提案 最终要达成一致的值 value 就在提案里



问题描述

假设有一组可以**提出（propose）value**（value 在提案 Proposal 里）的**进程集合**。一个一致性算法需要保证提出的多个 value 中，**只有一个** value 被选定（chosen）。如果没有 value 被提出，就不应该有 value 被选定。如果一个 value 被选定，那么所有进程都应该能**学习（learn）**到这个被选定的 value。对于一致性算法，**安全性（safaty）**要求如下：

- 只有被提出的 value 才能被选定。
- 只有一个 value 被选定，并且
- 如果某个进程认为某个 value 被选定了，那么这个 value 必须是真的被选定的那个。

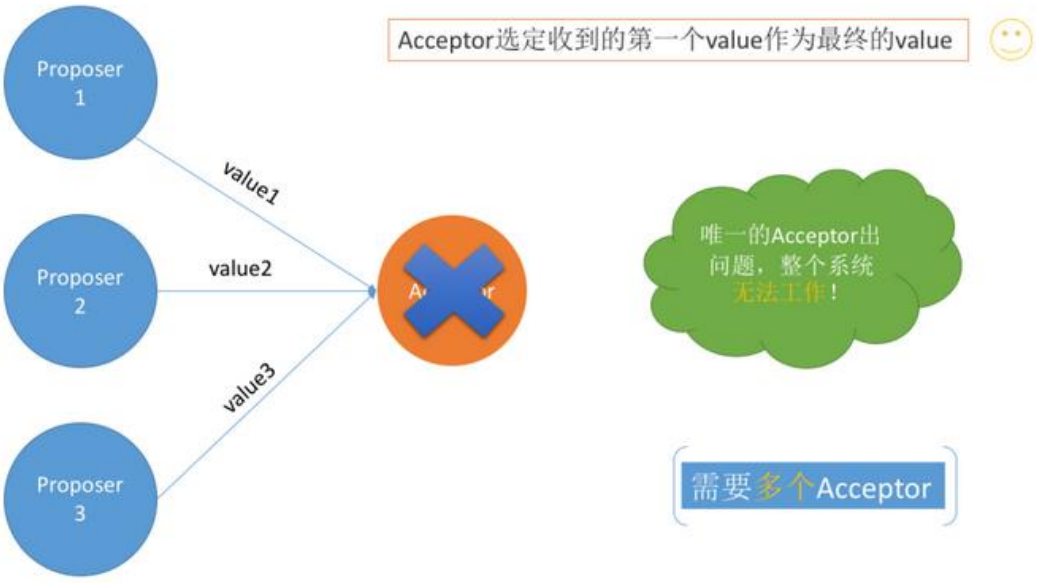
Paxos 的目标：保证最终有一个 value 会被选定，当 value 被选定后，进程最终也能获取到被选定的 value。

Choosing a Value（选定值）

最简单的方案——只有一个 Acceptor

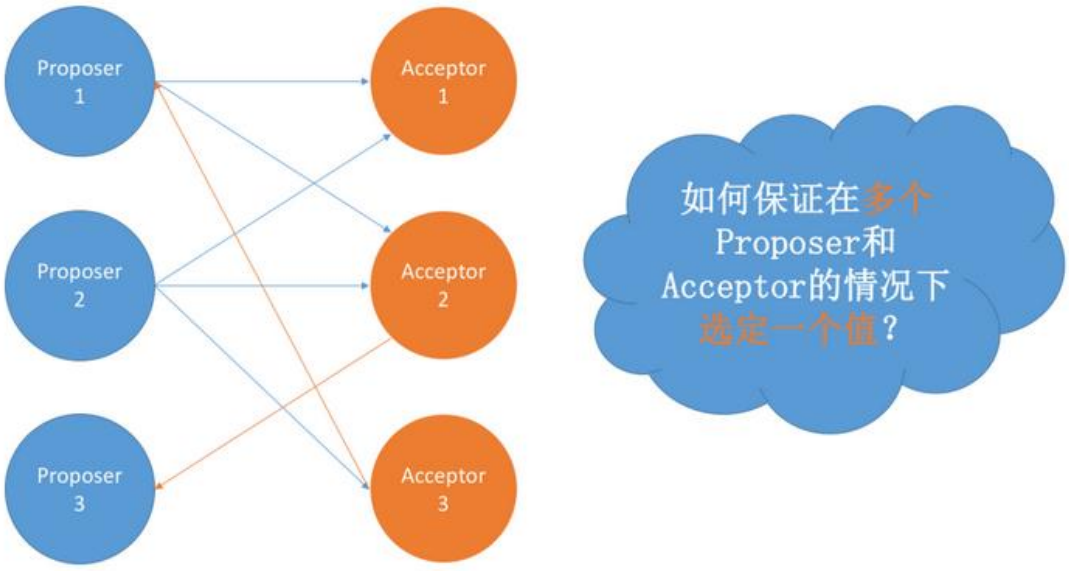
假设只有一个 Acceptor（可以有多个 Proposer），只要 Acceptor 接受它收到的第一个提案，则该提案被选定，该提案里的 value 就是被选定的 value。这样就保证只有一个 value 会被选定。

但是，如果这个唯一的 Acceptor 宕机了，那么整个系统就**无法工作**了！



多个 **Acceptor**

超过半数的 Acceptor 接受了一个议案，那我们就可以说这个提案被选定了（Chosen）



P1：一个 Acceptor 必须接受它收到的第一个提案。
规定：一个提案被选定需要被**半数以上**的 Acceptor 接受

多个提案
给每个提案一个全局编号,提案=[编号，值] ([n,v])
允许一个 acceptor 接受多个提案，后接受的提案
可以覆盖掉之前接受的提案。
P2：如果某个 value 为 v 的提案被选定了，那么每个编号更高的被选定提案的 value 必须

也是 v。
P2a：如果某个 value 为 v 的提案被选定了，那么每个编号更高的被 Acceptor 接受的提案的 value 必须也是 v。
P2b：如果某个 value 为 v 的提案被选定了，那么之后任何 Proposer 提出的编号更高的提案的 value 必须也是 v。
P2c：对于任意的 N 和 V，如果提案[N, V]被提出，那么存在一个半数以上的 Acceptor 组成的集合 S，满足以下两个条件中的任意一个：

- S 中每个 Acceptor 都没有接受过编号小于 N 的提案。
- S 中 Acceptor 接受过的最大编号的提案的 value 为 V。

提案生成算法

1. Proposer 选择一个**新的提案编号 N**，然后向**某个 Acceptor 集合**（半数以上）发送请求，要求该集合中的每个 Acceptor 做出如下响应（response）。 (a) 向 Proposer 承诺保证**不再接受**任何编号**小于 N 的提案**。
   (b) 如果 Acceptor 已经接受过提案，那么就向 Proposer 响应**已经接受过**的编号小于 N 的**最大编号的提案**。
   我们将该请求称为**编号为 N 的 Prepare 请求**。
2. 如果 Proposer 收到了**半数以上**的 Acceptor 的**响应**，那么它就可以生成编号为 N，Value 为 V 的**提案[N,V]**。这里的 V 是所有的响应中**编号最大的提案的 Value**。如果所有的响应中**都没有提案**，那 么此时 V 就可以由 Proposer **自己选择**。
   生成提案后，Proposer 将该**提案**发送给**半数以上**的 Acceptor 集合，并期望这些 Acceptor 能接受该提案。我们称该请求为 **Accept 请求**。（注意：此时接受 Accept 请求的 Acceptor 集合**不一定**是之前响应 Prepare 请求的 Acceptor 集合）

P1a：一个 Acceptor 只要尚**未响应过**任何**编号大于 N 的 Prepare 请求**，那么他就可以**接受**这个**编号为 N 的提案**。

Paxos 算法

阶段一：

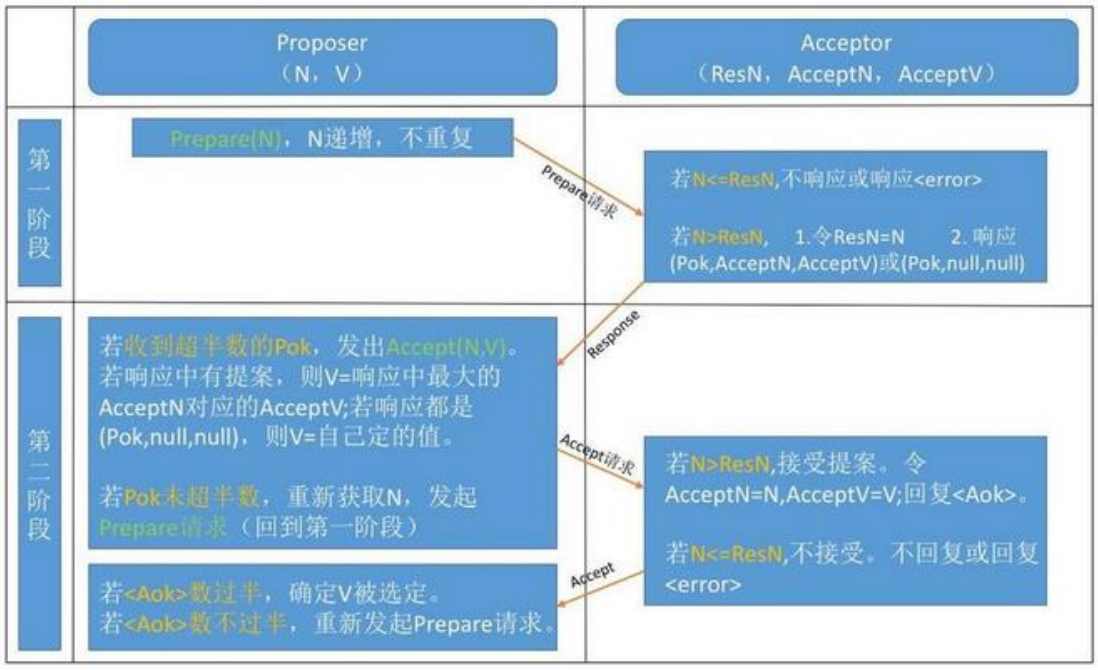
(a) Proposer 选择一个**提案编号 N**，然后向**半数以上**的 Acceptor 发送编号为 N 的 **Prepare 请求**。

(b) 如果一个 Acceptor 收到一个编号为 N 的 Prepare 请求，且 N **大于**该 Acceptor 已经**响应过的**所有 **Prepare 请求**的编号，那么它就会将它已经**接受过的编号最大的提案（如果有的话）**作为响应反馈给 Proposer，同时该 Acceptor 承诺**不再接受**任何**编号小于 N 的提案**。

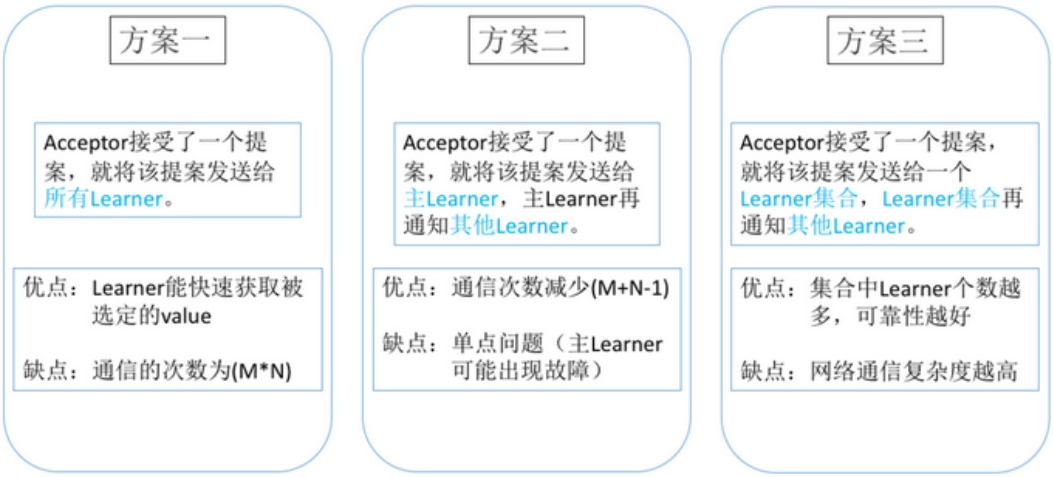
阶段二：

(a) 如果 Proposer 收到**半数以上** Acceptor 对其发出的编号为 N 的 Prepare 请求的**响应**，那么它就会发送一个针对**[N,V]提案**的 **Accept 请求**给**半数以上**的 Acceptor。注意：V 就是收到的**响应**中**编号最大的提案的 value**，如果响应中**不包含任何提案**，那么 V 就由 Proposer **自己决定**。

(b) 如果 Acceptor 收到一个针对编号为 N 的提案的 Accept 请求，只要该 Acceptor **没有**对编号**大于 N** 的 **Prepare 请求**做出过**响应**，它就**接受该提案**。



Learner 学习（获取）被选定的 value 有如下三种方案：



Paxos 算法的活性

Raft

Raft 是一种用来管理日志复制的一致性算法

优势

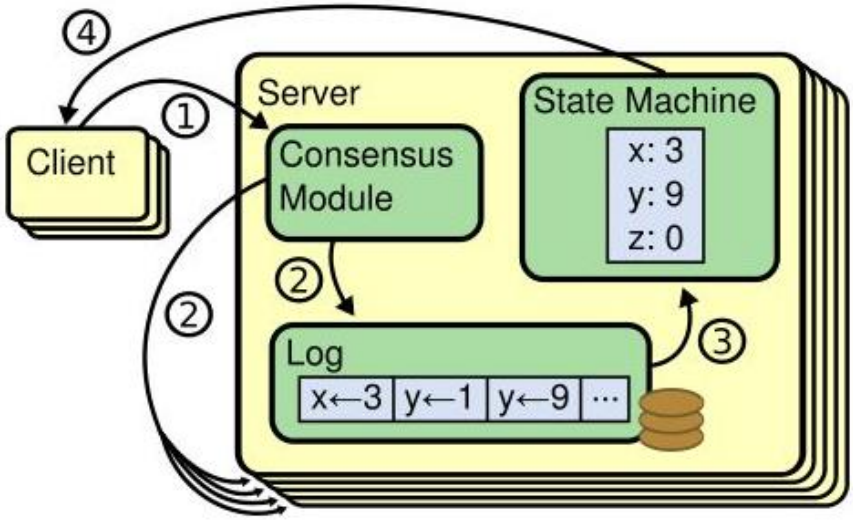
- **比其他算法更简单、更容易理解**
- **能满足一个实际系统的需求**

- **拥有许多开源的实现并且被许多公司所使用**
- **安全特性已经被证明**
- **效率和其他算法相比也具有竞争力**

复制状态机

一致性算法在复制状态机的背景下提出来的，**在这个方法中，一组服务器的状态机产生同样的状态副本，即使有一些服务器崩溃也能保证这组服务器继续执行**

架构如下：



1. 客户端向服务发起请求，执行指定操作
2. 共识模块将该操作以日志的形式备份到其他备份实例上
3. 当日志安全备份后，指定操作被应用于上层状态机
4. 服务器返回操作结果至客户端

在复制状态机中，分布式共识算法的职责就是按照**固定的顺序**将指定的日志内容备份到集群的其他实例上。

一致性算法

任务：保证复制日志一致

特性：

- **确保安全性：**从来不会返回一个错误的结果。
- **高可用性：**只要集群中的部分机器都能运行，可以互相通信并且可以和客户端通信，这个集群就可用。（>=2n+1)
- **不依赖时序保证一致性：**始终错误和极端情况下的信息延迟在最坏的情况下才会引起可用性问题。

Raft 目标

它必须提供一个完整的、实际的基础来进行系统构建，减少开发者的工作；

它必须在所有情况下都能保证安全可用；

对于常规操作必须高效；

易于理解，它必须使得大多数人能够很容易的理解；

必须能让开发者有一个直观的认识

Raft 改进方法

问题分解

- 领导人选取（Leader election）
- 日志复制（Log replication）
- 安全性（Safety）

状态空间简化

- 使整个系统更加一致，尽可能消除不确定性
- 日志之间不允许存在空洞，Raft 限制了日志不一致的可能性
- 使用随机化简化 Raft 中的领导选取算法
    - 随机化方法使得不确定性增加，减少了状态空间

Raft 特性

- **强领导者（Strong Leader）：**

Raft 使用一种比其他算法更强的领导形式。例如，日志条目从领导者发送向其他服务器，从而简化了对日志复制的管理，使得 Raft 更易于理解。

- **领导选取（Leader Selection）：**

Raft 使用随机定时器来选取领导者。（选举超时 Election timeout）

- **成员变更（Membership Change）：**

Raft 为了调整集群中成员关系使用了新的联合一致性（joint consensus）的方法，这种方法中大多数不同配置的机器在转换关系的时候会交迭（overlap）。这使得在配置改变的时候，集群能够继续操作。

Raft 通过首先选举一个领导人来实现一致性，然后给予领导人完全管理复制日志（**replicated log**）的权限。

领导人接收来自客户端的日志条目，并把它们备份到其他的服务器上，领导人还要告诉服务器们什么时候将日志条目应用到它们的状态机是安全的。通过选出领导人能够简化复制日志的管理工作。

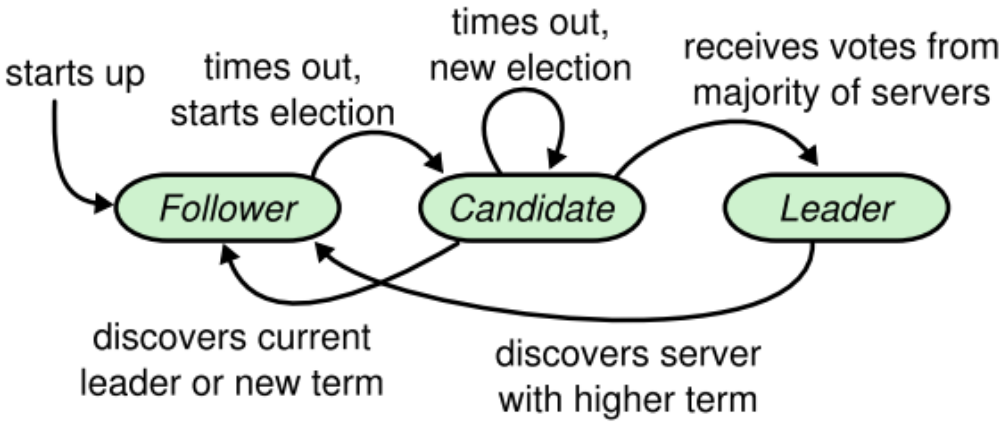
例如，领导人能够决定将新的日志条目放到哪，而并不需要和其他的服务器商议，数据流被简化成从领导人流向其他服务器。

如果领导人宕机或者和其他服务器失去连接，则选举下一个领导人。

三种角色

- **Leader** 负责从客户端处接收新的日志记录，备份到其他服务器上，并在日志安全备份后，通知其他服务器将该日志记录应用到位于其上层的状态机上
- **Follower** 总是处于被动状态，负责响应来自 **Leader** 或 **Candidate** 的请求，而自身不会发出任何请求
- **Candidate** 在 **Leader** 选举时，负责投票选出 **Leader**

角色转换



- 所有节点初始状态都是 Follower 角色
- Follower 超时时间内没有收到 Leader 的请求则转换为 Candidate 进行选举
- Candidate 收到大多数节点的选票则转换为 Leader；发现 Leader 或者收到更高任期的请求则转换为 Follower
- Leader 在收到更高任期的请求后转换为 Follower

任期

- Raft 把时间切割为任意长度的任期，每个任期都有一个任期号，采用连续的整数。

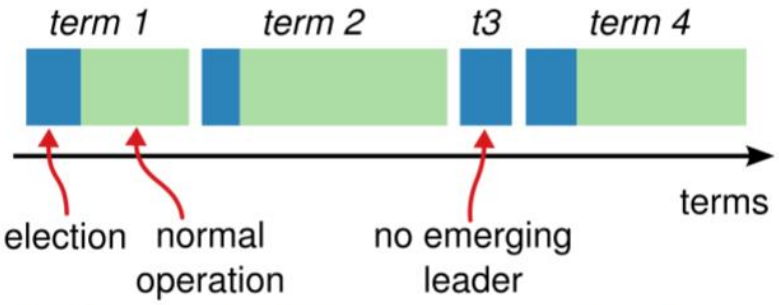


Figure 5: Time is divided into terms, and each term begins with an election. After a successful election, a single leader manages the cluster until the end of the term. Some elections fail, in which case the term ends without choosing a leader. The transitions between terms may be observed at different times on different servers.

每个任期都由一次选举开始，若选举失败则这个任期内没有 Leader；如果选举出了 Leader 则这个任期内有 Leader 负责集群状态管理。

Raft 中的服务器通过远程过程调用（RPC）来通信，基本的 Raft 一致性算法仅需要 2 种 RPC。

- RequestVote RPC 候选人在选举过程中触发
- AppendEntries RPC 领导人触发的，目的是复制日志条目和提供一种心跳（heartbeat）机制

Raft 使用一种心跳机制（heartbeat）来触发领导人的选取。

Heartbeat：不带有任何日志条目的 AppendEntries RPC

如果一个追随者在一个周期内没有收到心跳信息，就叫做选举超时

Leader 发送 Heartbeat 以保持所有节点的状态，Follower 收到 Leader 的 Heartbeat 后重设 Timeout。

只有超过一半的节点投票支持，Candidate 才会被选举为 Leader

为了防止在一开始是选票就被瓜分，选举超时时间是在一个固定的间隔内随机选出来的（例如，150~300ms）

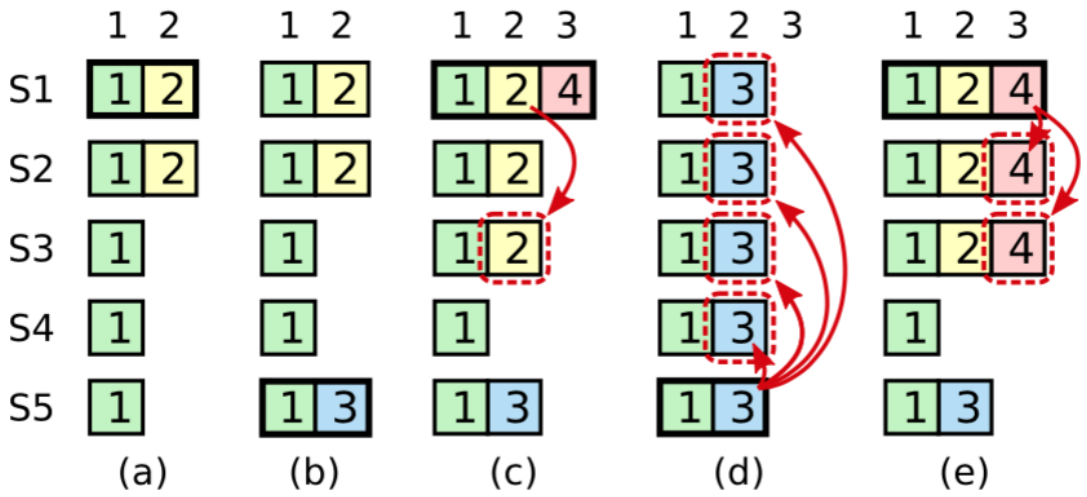
领导人在一个任期里在给定的一个日志索引位置最多创建一条日志条目，同时该条目在日志中的位置不会改变。

日志流向：从领导人流向追随者，领导人永远不会覆盖已经存在的日志条目。

Raft 通过比较日志中最后一个条目的索引和任期号来决定两个日志哪一个更新。

- **日志的任期号不同，任期号大的更新；**
- **日志的任期号相同，索引更长的日志更新。**

日志提交限制



上图按时间序列展示了 Leader 在提交日志时可能会遇到的问题。

1. 在 (a) 中，S1 是领导者，部分的复制了索引位置 2 的日志条目。
2. 在 (b) 中，S1 崩溃了，然后 S5 在任期 3 里通过 S3、S4 和自己的选票赢得选举，然后从客户端接收了一条不一样的日志条目放在了索引 2 处。
3. 然后到 (c)，S5 又崩溃了；S1 重新启动，选举成功，开始复制日志。在这时，来自任期 2 的那条日志已经被复制到了集群中的大多数机器上，但是还没有被提交。
4. 如果 S1 在 (d) 中又崩溃了，S5 可以重新被选举成功（通过来自 S2，S3 和 S4 的选票），然后覆盖了他们在索引 2 处的日志。反之，如果在崩溃之前，S1 把自己主导的新任期里产生的日志条目复制到了大多数机器上，就如 (e) 中那样，那么在后面任期里面这些新的日志条目就会被提交（因为 S5 就不可能选举成功）。 这样在同一时刻就同时保证了，之前的所有老的日志条目就会被提交。

任期 2 内产生的日志可能在(d)的情况下被覆盖，所以在出现(c)的状态下，Leader 节点是不能 commit 任期 2 的日志条目的，即不能更新 commitIndex。

在上图最终状态是(e)的情况下，commitIndex 的变化应该是 1->3，即在(c)的情况下，任期 4 在索引 3 的位置 commit 了一条消息，commitIndex 直接被修改成 3。

而任期 2 的那条日志会通过 Log Matching Property 最终被复制到大多数节点企且被应用。

成员变更

为了保证安全性，Raft 采用了一种两阶段的方式。

第一阶段称为 joint consensus，当 joint consensus 被提交后切换到新的配置下。

joint consensus 状态下：

- 日志被提交给新老配置下所有的节点
- 新旧配置中所有机器都可能称为 Leader
- 达成一致（选举和提交）要在两种配置上获得超过半数的支持

性质

- **Election Safety**（选举安全）：在任意给定的 **Term** 中，至多一个节点会被选举为 **Leader**
- **Leader Append-Only**（**Leader** 只追加）：**Leader** 绝不会覆盖或删除其所记录的日志，只会追加日志
- **Log Matching**（日志匹配）：若两份日志在给定 **Term** 及给定 **index** 值处有相同的记录，那么两份日志在该位置及之前的所有内容完全一致
- **Leader Completeness**（**Leader** 完整性）：若给定日志记录在某一个 **Term** 中已经被提交，那么后续所有 **Term** 的 **Leader** 都将包含该日志记录
- **State Machine Safety**（状态机安全性）：如果一个服务器在给定 **index** 值处将某个日志记录应用于其上层状态机，那么其他服务器在该 **index** 值处都只会应用相同的日志记录