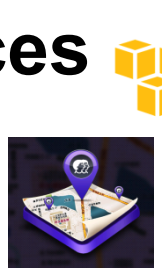
服务计算
Service Computing

王旭
http://xuwang.tech

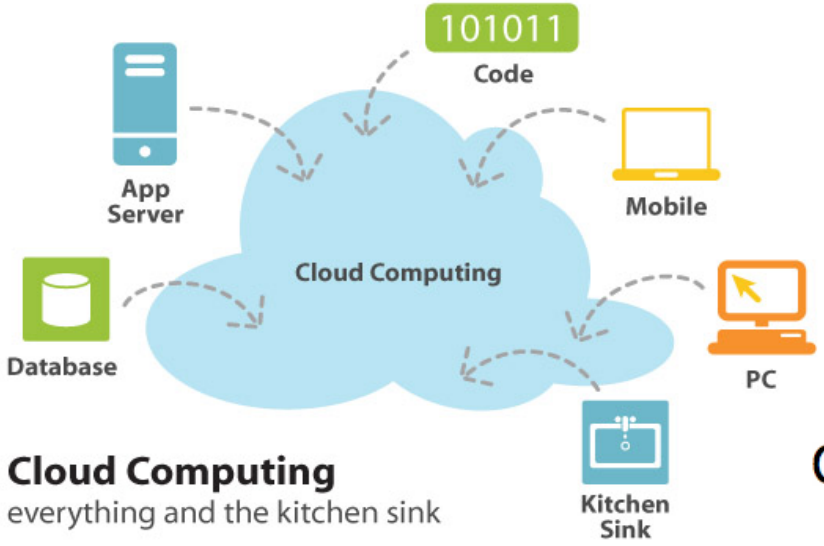
八、基于副本的服务质量保障

在线服务

- 常见的互联网软件服务
  - Facebook的社会网络服务
  - Google的网页搜索服务
  - 淘宝网的电子商务服务
- 云计算: XaaS, Internet of Services
  - Amazon的计算和存储服务
- 移动计算：位置服务LBS
- 在线服务已经成为了软件的重要形态和未来的发展趋势



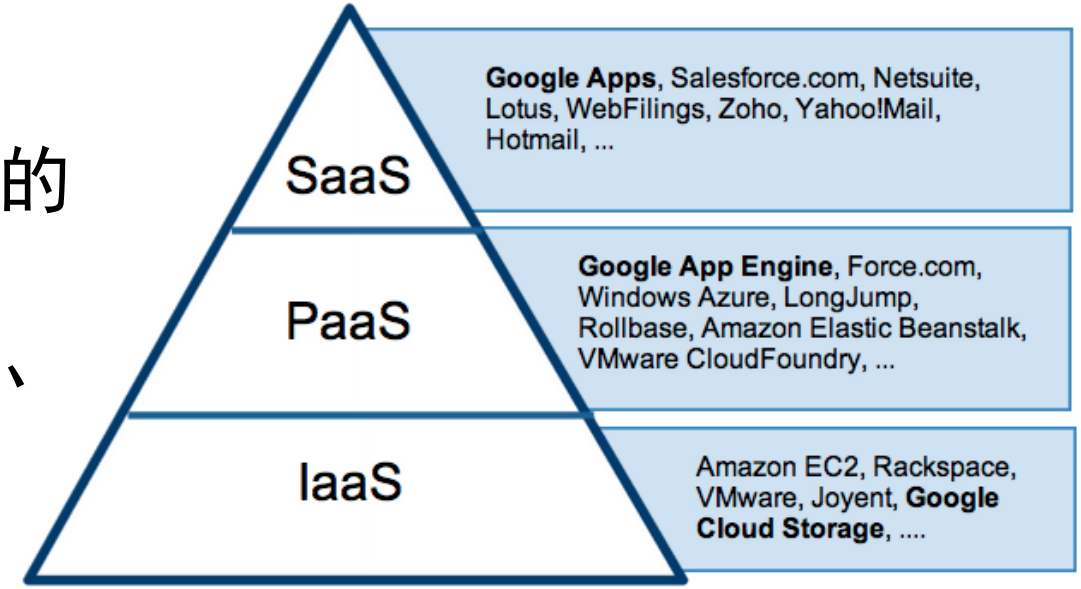
云计算是“服务”的一种实现



Cloud Computing as Gartner Sees It

传统服务计算：服务的建模、设计与开发

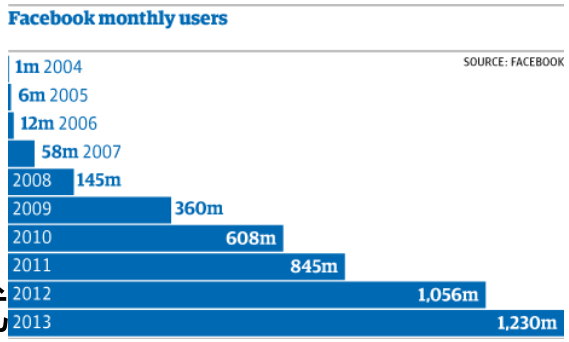
云计算：服务的运行、提供与访问



Source: Gartner AADI Summit Dec 2009

可扩容性

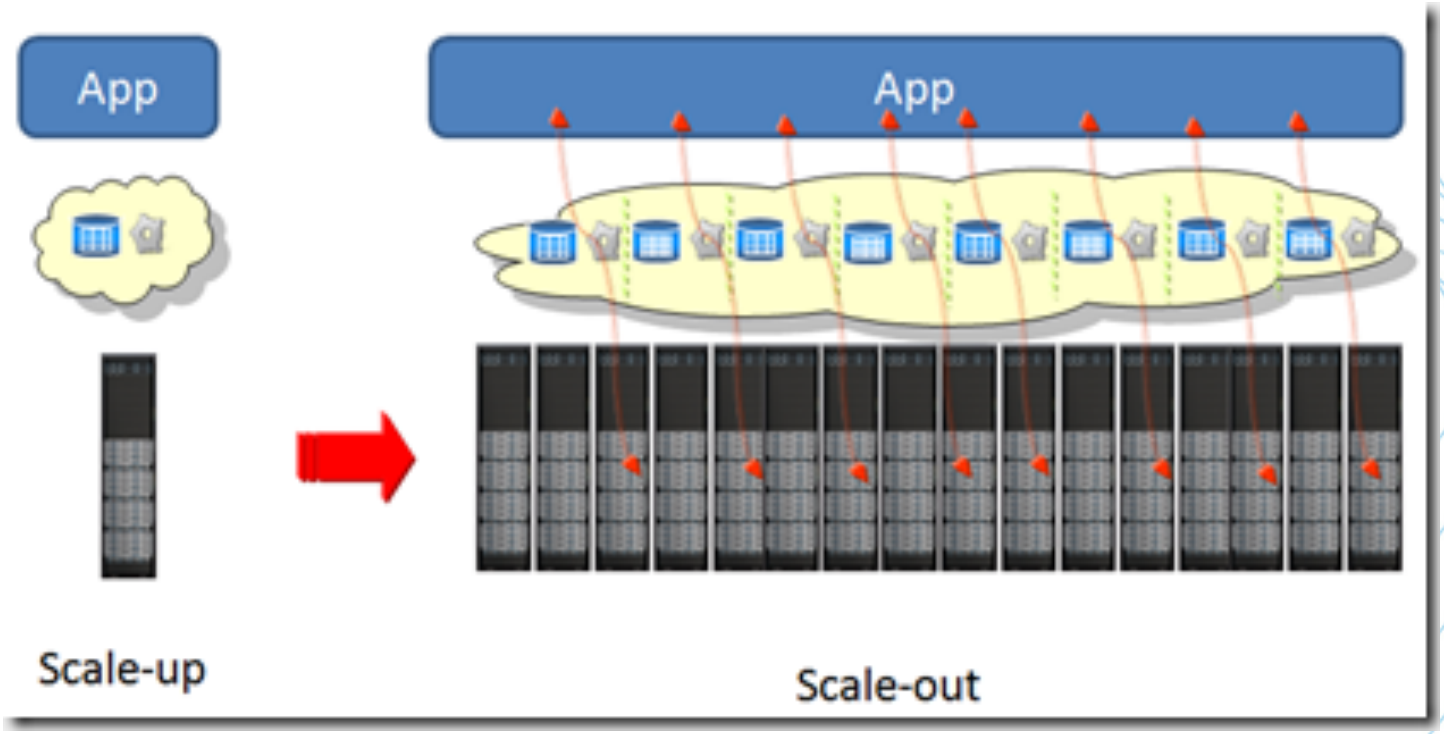
- **软件服务的可扩容性(scalability)**
  - **用户量大**
    - Facebook用户数突破十二亿
  - **数据量大**
    - 2013年左右Google每天处理的数据就
  - **并发访问量大**
    - 2012年11月11日淘宝网促销活动第一分钟超过1000万用户访问，高峰成交量19.2万/分钟
    - 2014年交易额增长到3倍
- **高可扩容性成为了软件服务的重要需求**



云计算支撑服务的大规模访问

- twitter — 13 billion API calls / day *(May 2011)*
- NETFLIX — 10 billion API calls / month *(January 2011)*
- amazon web services — Over 260 billion objects stored in S3 *(January 2011)*
- npr — 1.6 billion API-delivered stories / month *(October 2010)*
- Google — 5 billion API calls / day *(April 2010)*
- facebook — 5 billion API calls / day *(October 2009)*
- ebay — 8 billion API calls / month *(Q3 2009)*
- bing — 3 billion API calls / month *(March 2009)*

Scalability：可扩容性



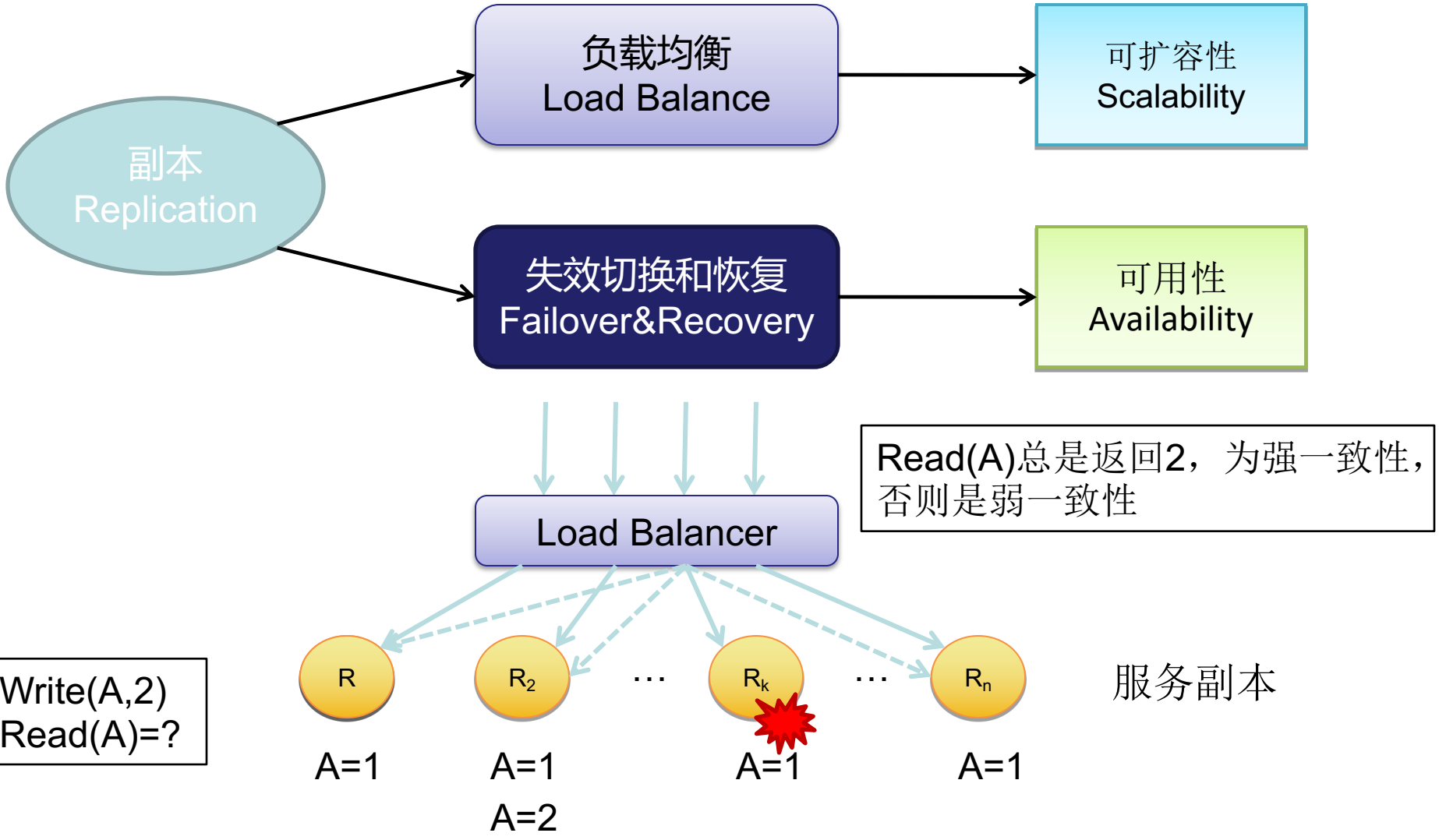
纵向扩展　　横向扩展

可用性

- **软件服务的可用性(availability): 7X24服务**
  - **服务等级约定(SLA):可用性**
    - Amazon EC2 99.95%, Google Cloud Storage 99.9%, Microsoft Windows Azure 99.9%
  - **大型互联网软件服务：宕机事件频发**

序号	公司	案例
1	Amazon	2011年4月21、22日，亚马逊EC2服务中断持续两天,并出现扩散，2012年6月和10月同样出现了宕机事件
2	微软	2012年7月26日，Azure故障，导致西欧用户受影响。此次中断持续2.5小时。
3	Google	GAE数据中心由google管理，中断时间是2012年10月26日，持续4小时
4	微软	微软Hotmail、Outlook.com以及SkyDrive等服务自美国2013年3月12日下午起，经历了一场长达17个小时的宕机事件。
5	Twitter	2013年6月3日，用户无法发布和读取tweets，持续大约45分钟
6	Google	2013年8月17日， 几乎大部分的Google服务出现宕机，导致全球互联网流量下降接近40%，持续几分钟
7	Adobe	2014年5月25日，Adobe Creative Cloud无法访问接近24小时
8	微软	2014年6月23、24日，Lync instant messaging service、Outlook Exchange online相继出现故障

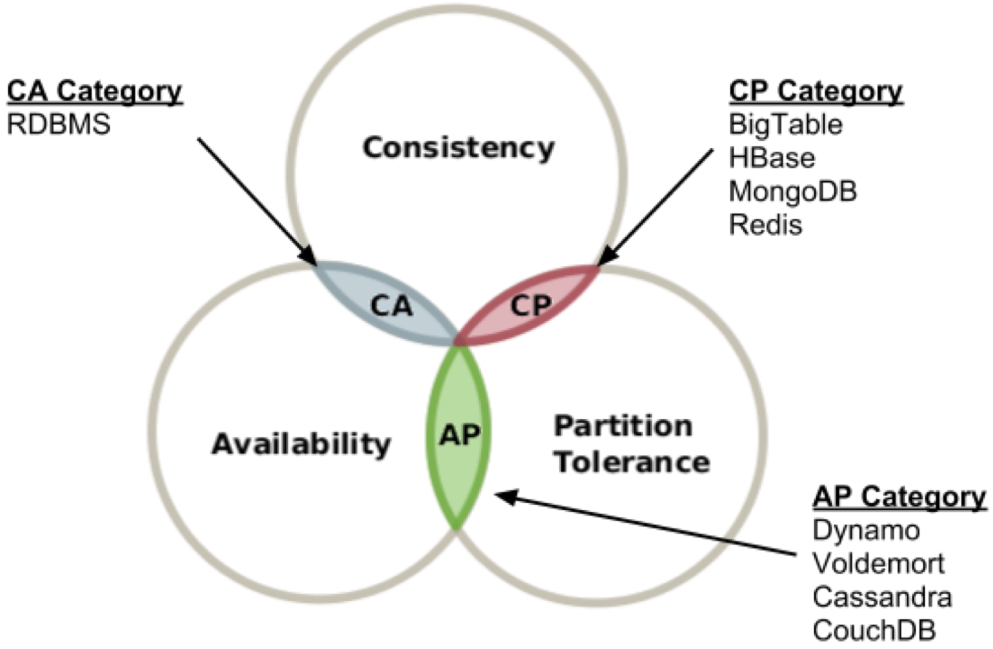
软件服务副本和(强/弱)一致性



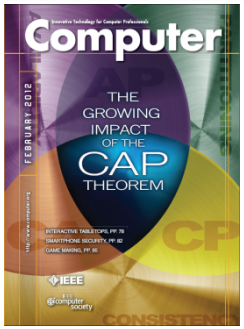
副本一致性(1)

- CAP(2000年SIGMOD会议keynote)
  - 一致性(Consistency)、可用性(Availability)和网络分割容忍性(Partition)之间最多只能两个同时满足
- PACELC(2012 IEEE Computer)
  - 如果副本节点间存在分割(Partition),可用性(Availability)和一致性(Consistency) 存在权衡；否则(Else)，延迟(Latency)和一致性(Consistency)之间存在权衡

Scalability: CAP理论



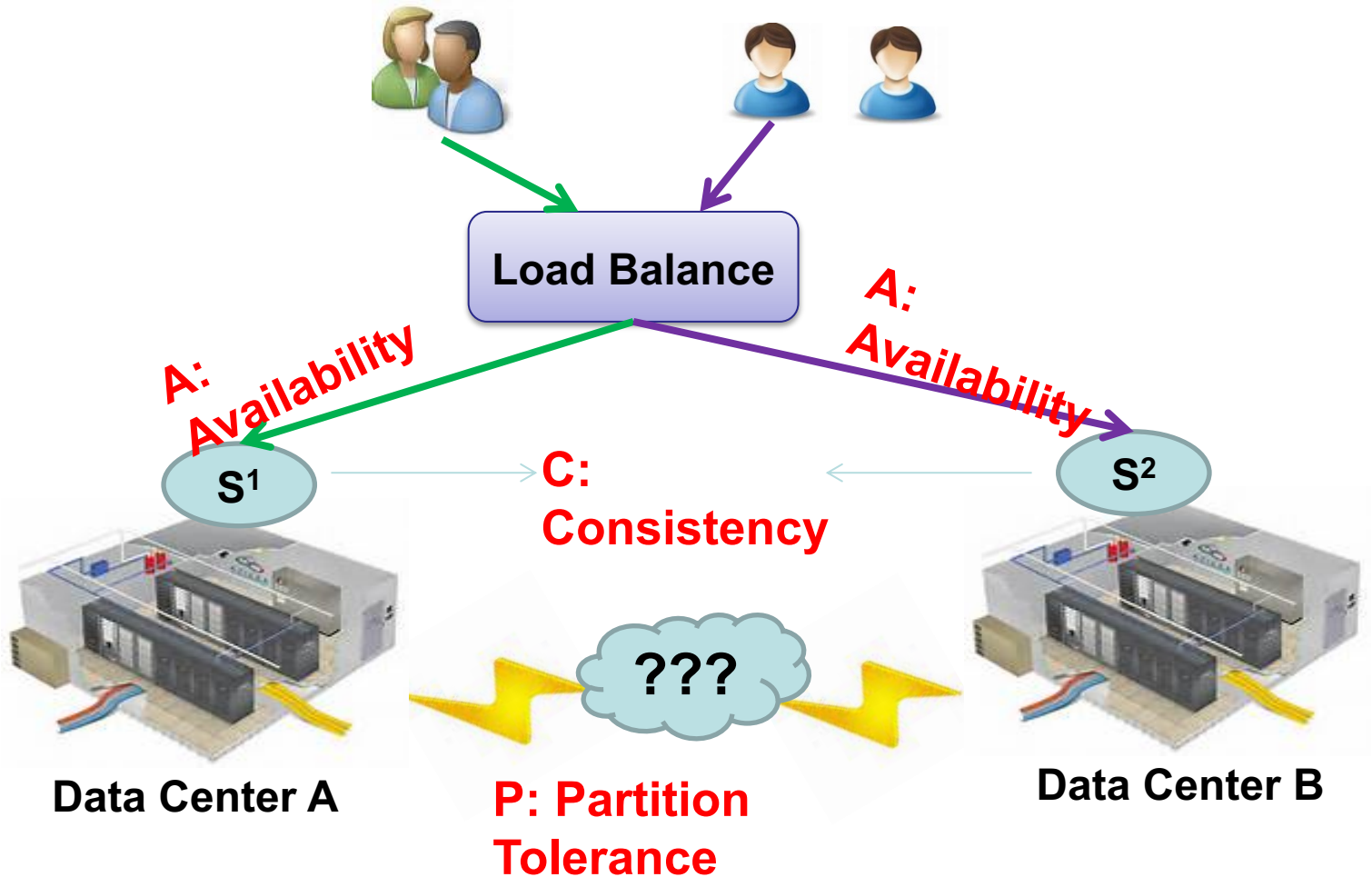
PODC 2000: C、A与P只能同时拥有其中两个属性！



2012年IEEE Computer 专刊：*The Growing Impact of the CAP Theorem*

相关术语：**ACID**, **BASE**, **PACELC**

约束：CAP三个属性



- **Consistency: all nodes have same data at any time**
- **Availability: the system allows operations all the time**
- **Partition-tolerance: the system continues to work in spite of network partitions**

副本一致性(2)

- 图灵奖获得者Jim Gray
  - 强一致性条件下副本节点（数据）的增长会导致副本间的竞争代价呈现三次方关系的增长
- Amazon的在线购物车服务
  - 总是可写
  - 最终一致性
- Yahoo的全球图片分享服务
  - 全球不同地区的图片数据强一致性同步代价将非常昂贵
  - 每隔一段时间再同步

重新思考一致性

- 传统一致性：安全性有关
  - 线性一致性(Linearizability)、 ACID
- 强一致性→弱一致性
  - 最终一致性(eventual consistency)
  - ACID→BASE(Basic Availability, Soft state, Eventual consistency)
- 弱一致性存在的问题
  - 写冲突(Write conflicts)，导致副本间的数据状态不一致性
  - 读陈旧(Read Staleness), 可能读到陈旧的数据

服务质量保障目标

- **弱一致性**
  - **尽可能的提升性能（可扩容性）**
    - 量化一致性和性能，寻找一致性和性能之间的优化权衡方法
  - **追求高可用性**
    - 量化一致性和可用性，寻找一致性和可用性之间权衡的优化权衡方法
- **强一致性**
  - **结合软件服务特征，强一致性条件下的副本性能优化**

副本协议类

- 主从备份
  - 一般主从两个副本节点，从节点进行备份
- 两阶段提交协议
  - 2PC，3PC
- 组通信
  - 协调多个分布式进程间的消息序
- Paxos协议：所有副本对消息序达成一致
- Quorum系统
  - 副本全集N，两个子集W(写请求)和R(读请求)

一致性模型

- 强一致性
  - 典型系统有SMATER、POSTGRES-R、HBase
- 弱一致性
  - 最终一致性（Eventual Consistency）
  - 时间一致性（Timeline consistency）
  - 会话一致性（Session consistency）
  - 因果一致性（Causal consistency）
  - 快照隔离一致性（Snapshot Isolation consistency）

服务质量保障方法

- 四个方面
  - 一致性和性能的权衡
  - 一致性和可用性的权衡
  - 强一致性副本协议的性能优化
  - 高可用的最强弱一致性：因果一致性

服务质量保障方法

- 四个方面
  - 一致性和性能的权衡
  - 一致性和可用性的权衡
  - 强一致性副本协议的性能优化
  - 高可用的最强弱一致性：因果一致性

一致性和性能的权衡

- **核心是建立（不）一致性的量化模型**
  - **K版本误差量化模型（K-Versions）**
    - 用版本号量化不一致性
  - **读最新数据的概率模型**
    - 用读到最新数据的概率度量一致性
  - **写误差模型（TACT）**
    - 用副本数值之间的差值量化不一致性
  - **陈旧读时间和概率模型（PBS）**
    - 用写请求之一段时间后读请求读到该写请求的概率量化一致性

一致性和可用性的权衡

- **采用部分Quorum系统来设计副本协议，追求高可用性，未涉及可用性的量化**
  - **Amazon的Dynamo系统**
  - **开源分布式数据库Cassandra**
  - **基于经验配置W,R值**
    - W,R 越小，可用性越高，一致性越弱
- **现有工作对Quorum系统可用性的量化局限在简单网络**
  - **全连接网络、环(Ring)状网络等**

强一致性副本协议的性能优化

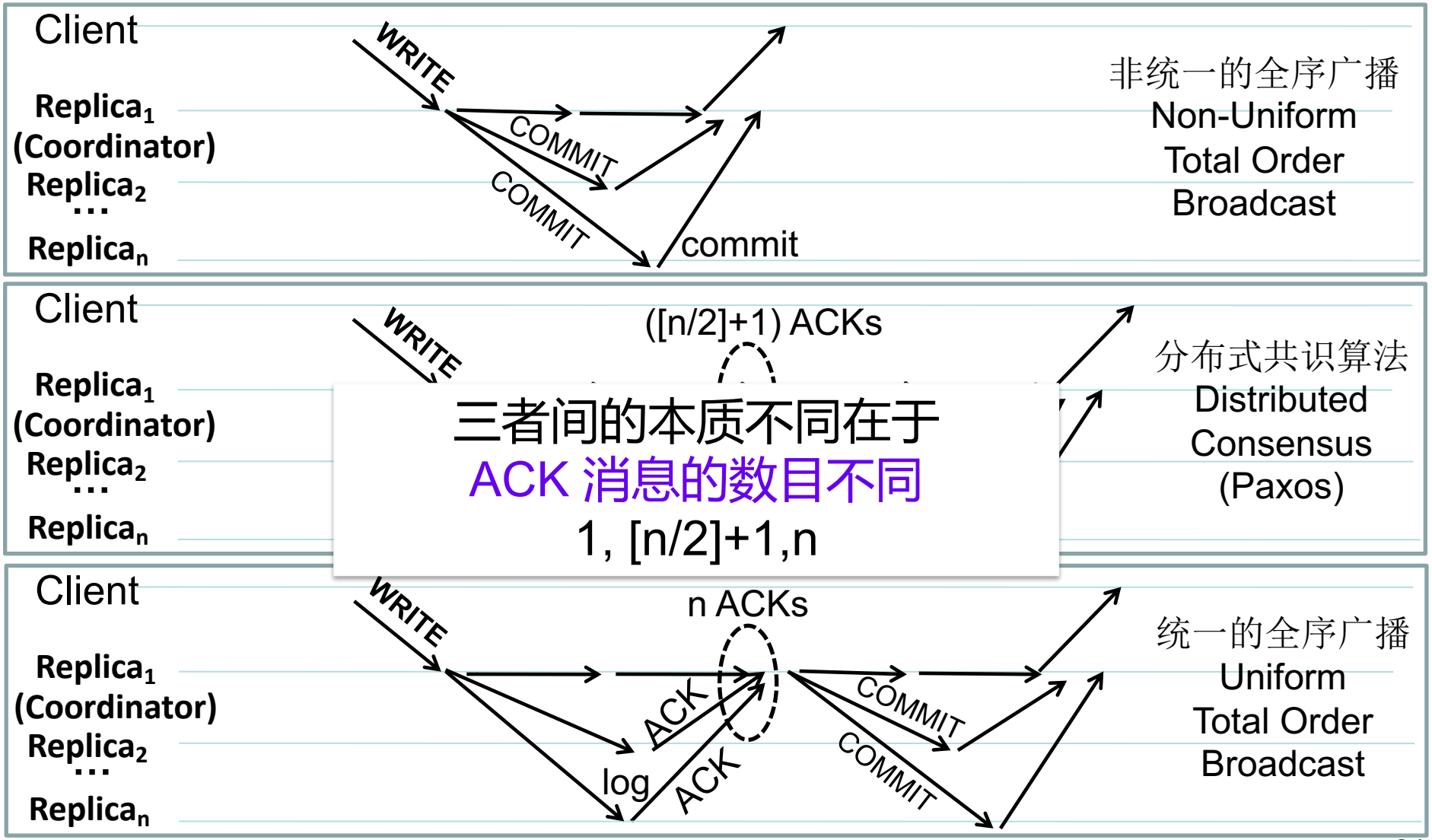
- 快速提交(Fast Paxos)
- 等价请求集合(Generalized Paxos)
- IP广播优化
- 流水线和批处理
- 多领导者分组提交(Mencius和Multi-Ring Paxos)
- 内存和SSD优化(Spinnaker和CORFU)

一致性和性能的权衡

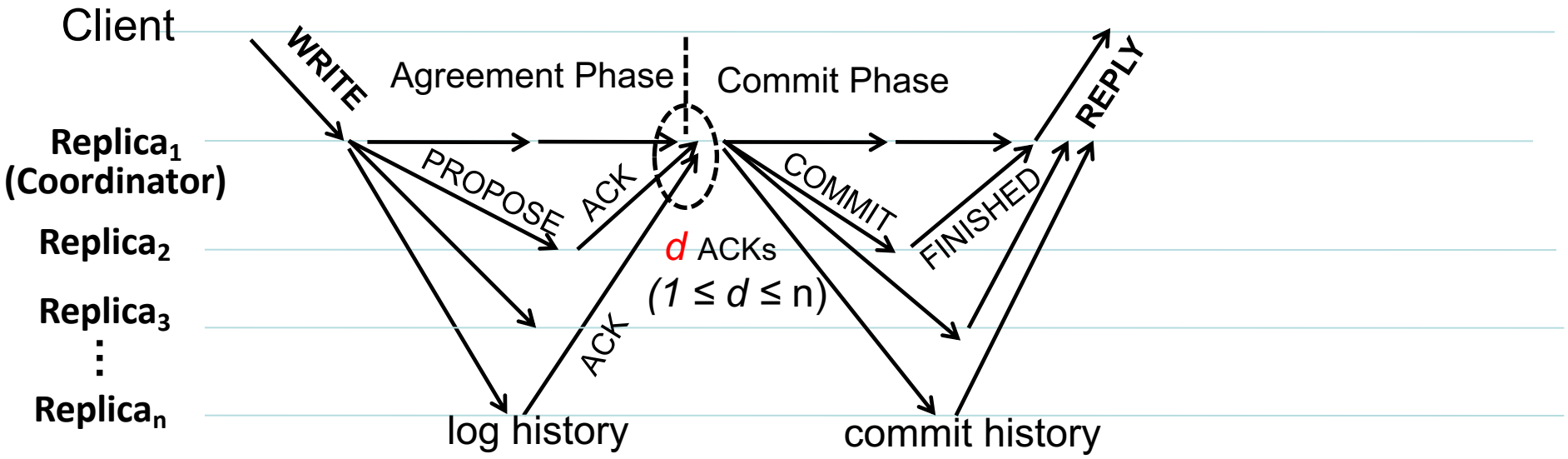
- 不一致性可能导致的问题
  - 读到陈旧数据
  - 写请求冲突，需要人工干预

- 延迟: 重要的系统性能属性
  - bing: +2s → 查询/用户 -1.8% and 收益/用户 -4.3%
  - Google: +500ms→ -25% 搜索次数
  - amazon: +100ms→ -1% 销售额

副本状态机的三种典型实现



系统模型: RSM-d

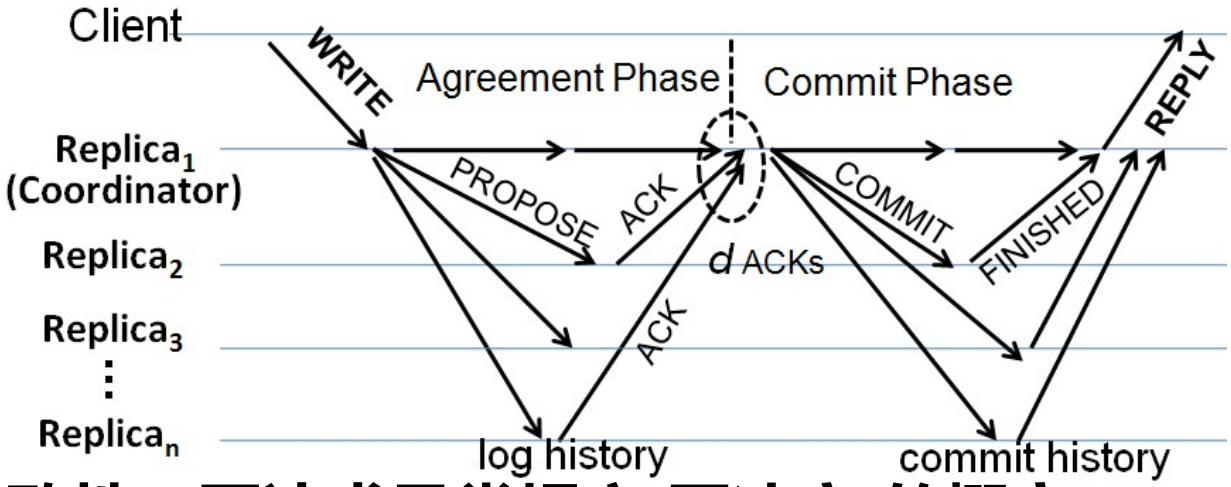


(1) *f* : 同时失效的节点数目，其中 $f \le [n/2]$ 保证问题可解

(2)一旦某个副本怀疑协调者(coordinator)失效, 它将基于日志历史(log history)和提交历史(commit history)选举出新的协调者并进行系统恢复.

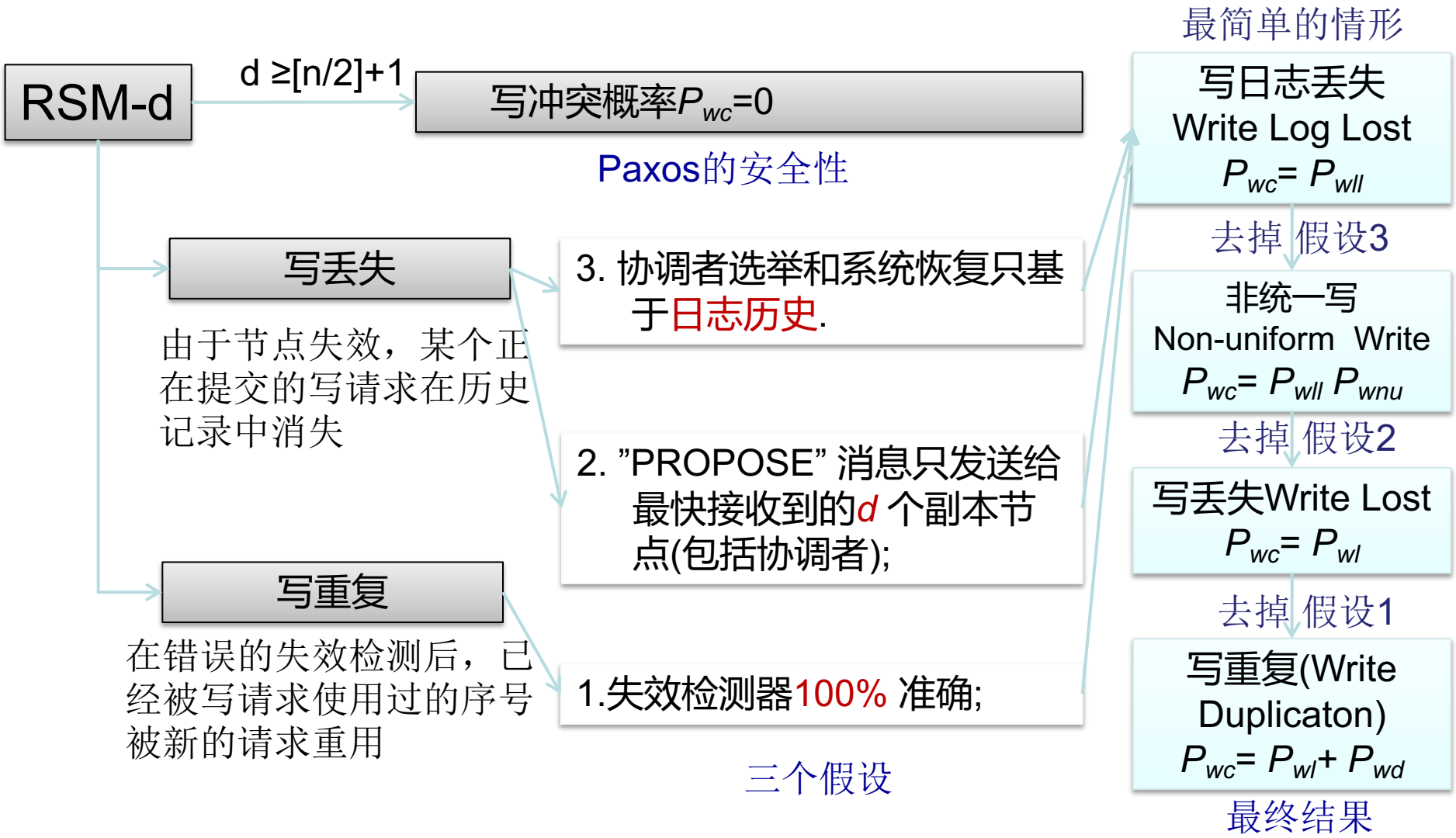
d=1	非统一全序广播(Non-Uniform Total Order Broadcast
d=[n/2]+1	Paxos
d=n	统一全序广播(Uniform Total Order Broadcast)
Otherwise	以前被忽略

一致性和性能的量化

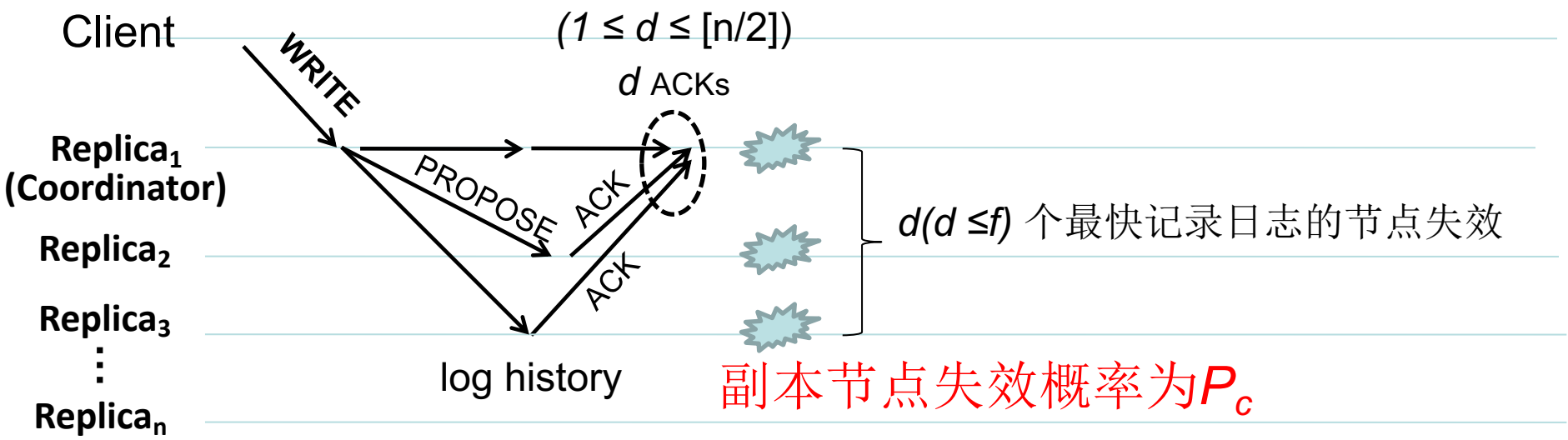


- 不一致性：写请求异常提交(写冲突)的概率
  - 写请求的历史记录都丢失(写丢失)
  - 写请求的协调节点被错误的认为失效(写重复)
  - 计算上述两类事件发生时写请求的异常提交概率
- 延迟
  - 建立从写请求提交到最终返回的整个处理过程
  - 计算整个过程的持续时间的期望值

一致性量化分析思路


RSM-d
d ≥[n/2]+1
写冲突概率P_{wc}=0
Paxos的安全性
写丢失
由于节点失效，某个正在提交的写请求在历史记录中消失
3. 协调者选举和系统恢复只基于日志历史.
2. ”PROPOSE” 消息只发送给最快接收到的d 个副本节点(包括协调者);
写重复
在错误的失效检测后，已经被写请求使用过的序号被新的请求重用
1.失效检测器100% 准确;
三个假设
最简单的情形
写日志丢失
Write Log Lost
$P_{wc}= P_{wll}$
去掉 假设3
非统一写
Non-uniform Write
$P_{wc}= P_{wll}\ P_{wnu}$
去掉 假设2
写丢失Write Lost
$P_{wc}= P_{wl}$
去掉 假设1
写重复(Write Duplicaton)
$P_{wc}= P_{wl}+ P_{wd}$
最终结果

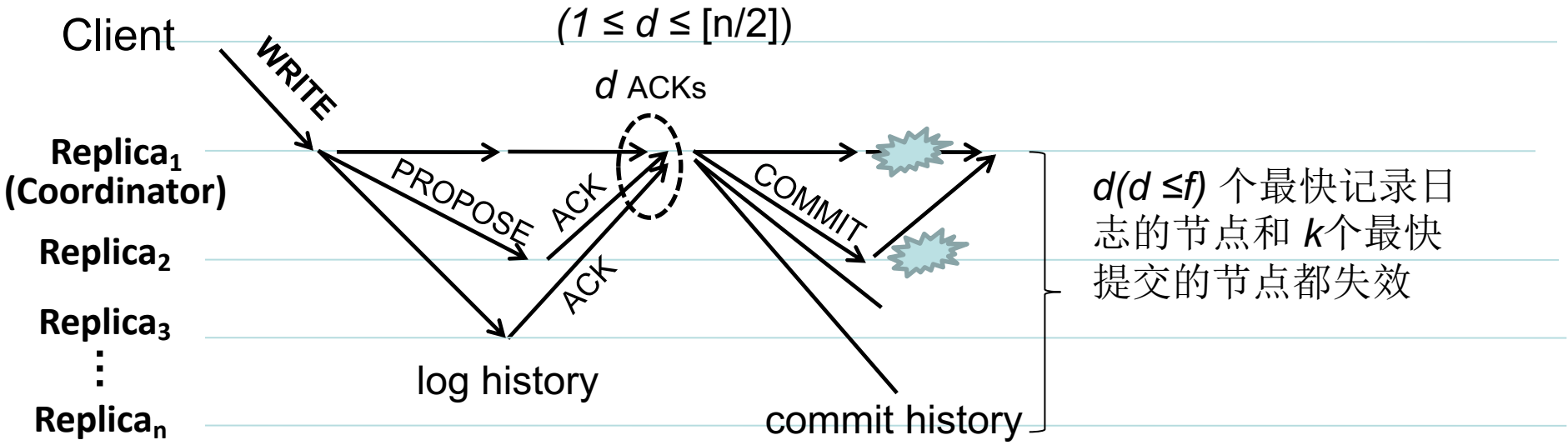
最简单的情形: 写日志丢失



- 假设 1,2,3
- ***d(d ≤f)*个最快记录日志的节点失效, 导致写请求丢失和写冲突的出现**
- ***P_{wll}*: 写请求在日志中丢失的概率**

$$P_{wc} = P_{wll} = (P_c)^d$$

非统一写(Non-uniform Write)



- **假设 1,2, 协调者选举和恢复时考虑提交历史(commit history)**
- ***P_{wnu}*** **: 写请求所有已提交的节点全部失效的概率**

$$P_{wnu} = \sum_{k=0}^{[n/2]-d} (P_c)^k (1-P_c)^{n-d-k}$$

k 为已提交节点的数目

$$P_{wc} = P_{wll} P_{wnu}$$

日志节点的增加

- 假设 1. 考虑异步传输的“PROPOSE”消息
- 随机变量D: 失效发生时日志记录的个数.
- $P_{elw}(D)$：D的累积密度函数
- 定义函数：

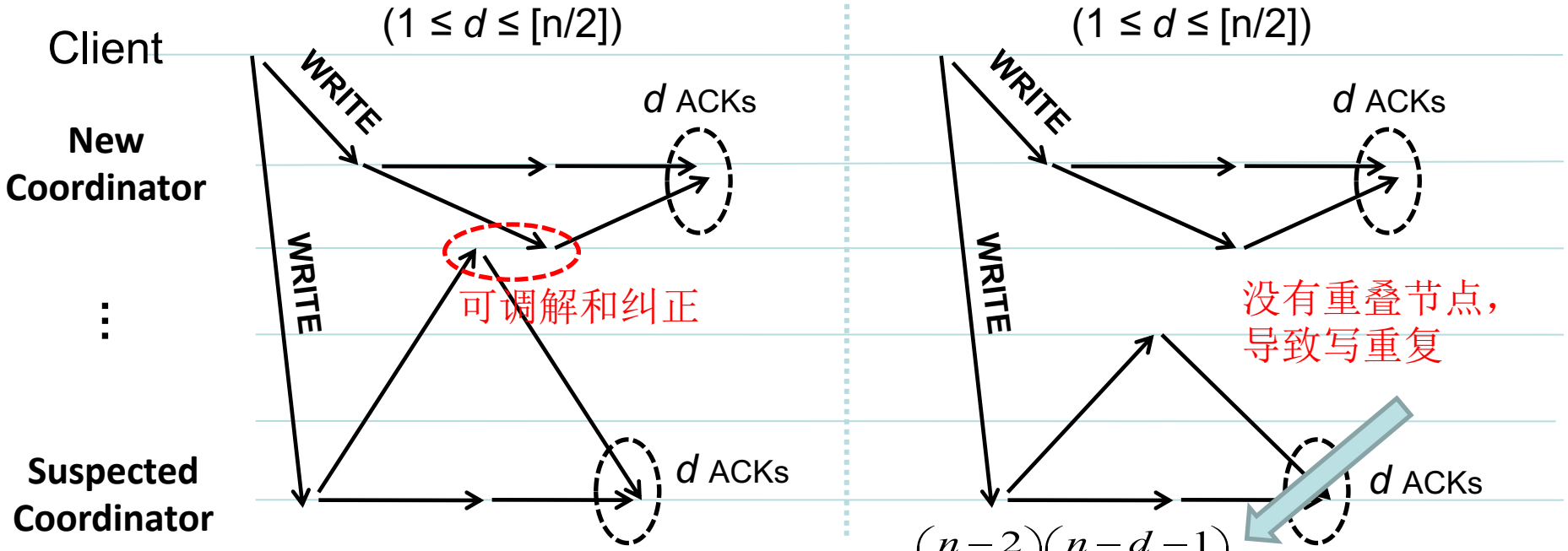
$$Q(x) = (P_c)^x \sum_{k=0}^{[n/2]-x} (P_c)^k (1-P_c)^{n-x-k}$$

- 如果 *D=d* (即假设1,2存在时), $P_{wc} = Q(d)$
- 类似地, 如果*D=m*(d ≤ m ≤ [n/2]) , 则有$P_{wc}(D=m) = Q(m)$
- 对任意D值的条件概率求和(也就是写请求丢失的概率P_{wl}):

$$P_{wc} = P_{wl} = \sum_{m=d}^{[n/2]} P_{elw}(m) Q(m)$$

写重复(Write Duplication)

- **三个假设都放弃**
  - **失效检测器有可能错误的怀疑协调者节点失效**
  - **新选举的协调者将会重复使用已经使用过的序号，导致写请求的重复**

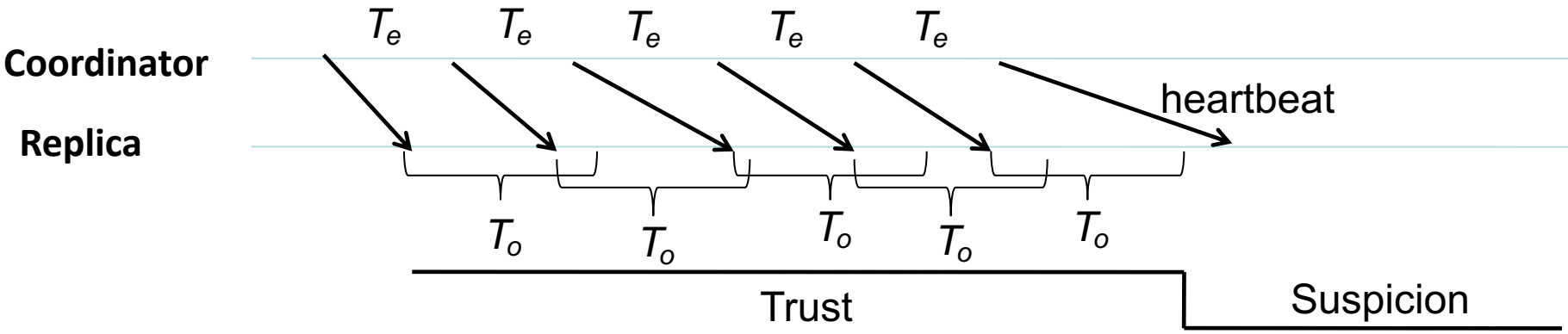


P_{no}：两个相同序号的写请求日志节点没有重叠的概率

$$P_{no} = \frac{\binom{n-2}{d-1}\binom{n-d-1}{d-1}}{\binom{n-1}{d-1}\binom{n-1}{d-1}}$$

失效检测器

- **常见的失效检测器**



T：副本节点间的消息延迟
$f(t)$: T的概率密度函数
T_{hb2} 和 T_{hb1} : T的两个样本.
P_{fs} : 一个副本错误怀疑协调者的概率

$$P_{fs} = \Pr(T_{hb2} + T_e > T_{hb1} + T_o) = \Pr(T_{hb2} - T_{hb1} > T_o - T_e)$$

第二个heartbeat消息超时

写重复的概率

- ***n*** **个副本中至少有一个错误怀疑协调者的概率：**

$$P_{gfs} = \sum_{f=0}^{[n/2]} \binom{n-1}{f} (P_c)^f (1-P_c)^{n-1-f} (1-(1-P_{fs})^{n-1-f})$$

- **写请求重复的概率:**

$$P_{wd} = (1-P_c) P_{gfs} P_{no}$$

协调者节点正常工作

至少一个副本错误怀疑协调者

两个相同序号的写请求日志节点不存在重叠

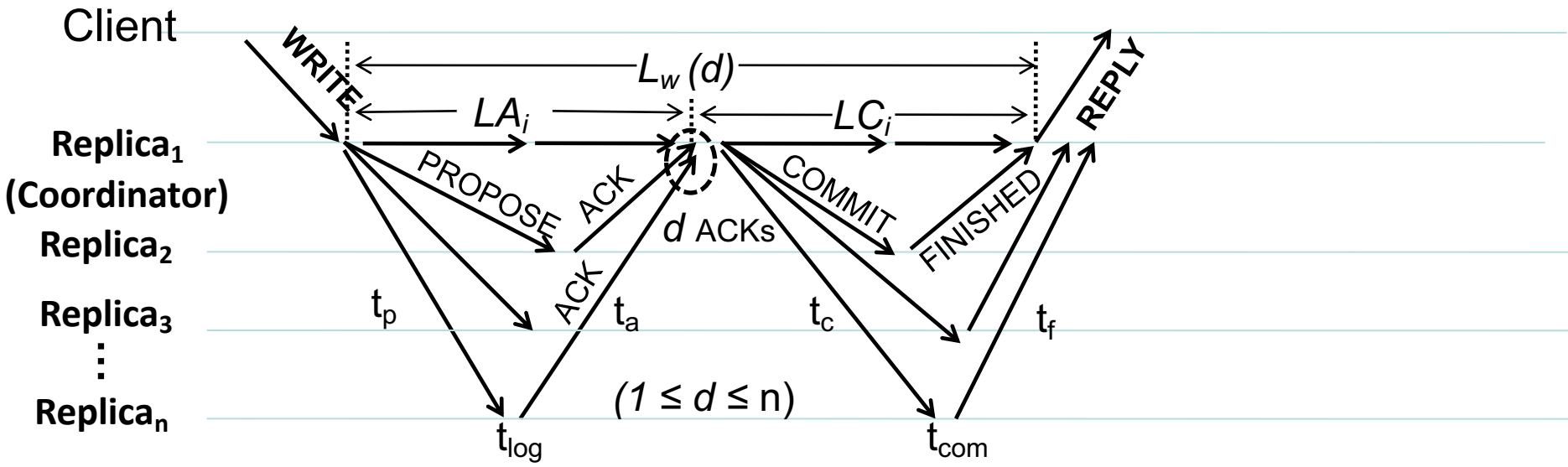
写冲突的概率

- 写丢失(Write lost)
  - 某些节点失效导致写请求的所有历史记录都丢失 (包括协调者)
- 写重复(Write duplication)
  - 当协调者正常工作但是被错误怀疑时发生

- 写丢失和写重复是互斥事件, 故:

$$P_{wc} = P_{wl} + P_{wd}$$

基于延迟的性能分析



$LA_i = t_p(i)+t_{log}(i)+t_a(i)$

将 $LA_1, LA_2, \ldots, LA_n$ 升序排列:

$$LA_{(1)} \le LA_{(2)} \le \ldots \le LA_{(n-1)}$$ (除开协调者)

Latency $L_w(d) = LA_{(d-1)}+min(LC_i)$

延迟$L_w(d)$的期望值

- $LA_i=t_p(i)+t_a(i)$ 概率密度函数$(t)=\int_{-\infty}^{+\infty} f(x)f(t-x)dx$ ，累积密度函数 $G(t)$
- $LA_{(k)}$ 概率密度函数和累积密度函数分别是$h_{(k)}(t)$和$H_{(k)}(t)$$(1 \le k \le n-1)$

kth 最小 $LA_i \in [t,t+\varepsilon]$的概率

一个$LA_i \in [t,t+\varepsilon]$

(k-1) 个LA_i 小于t

其余的 LA_i 大于 t+ ε

$$P(LA_{(k)} \in [t,t+\varepsilon]) = \sum_{i=1}^{n-1} P(LA_i \in [t,t+\varepsilon]) \binom{n-2}{k-1} P(LA<t)^{k-1} P(LA>t+\varepsilon)^{n-1-k}$$

$$h_{(k)}(t) = \lim_{\varepsilon \to 0} \frac{P(LA_{(k)} \in [t,t+\varepsilon])}{\varepsilon}$$

- 基于$h_{(k)}(t)$，可以求得:

$$E(LA_{(d)} - LA_{(d-1)}) = \int_{-\infty}^{+\infty} t h_{(d)}(t)dt - \int_{-\infty}^{+\infty} t h_{(d-1)}(t)dt$$

- 所以:

$$E(L_w(d+1)) - E(L_w(d)) = E(LA_{(d)} - LA_{(d-1)}) = \binom{n-1}{d-1} \Delta LA(d) > 0 (d \ge 1)$$

其中 $\Delta LA(d) = \int_0^{+\infty} (G(t))^{d-1}(1-G(t))^{n-d} dt > 0$

基于效益的一致性和延迟权衡

- 系统总效益$B(d)$，一致性正相关的效益B_c，延迟负相关的效益 B_l：

$$B_c = \alpha(1 - P_{wc}(d)) + \beta(\alpha > 0)$$
$$B_l = \gamma - \theta E(L_w(d))(\theta > 0)$$

- 于是:

$$\begin{aligned} B(d) &= B_c + B_l + \delta \\ &= \alpha(1 - P_{wc}(d)) + \beta + \gamma - \theta E(L_w(d)) + \delta \\ &= (\eta - \theta E(L_w(1))) - v(d) \end{aligned}$$

$$v(d) = \alpha P_{wc}(d) + \theta(E(L_w(d)) - E(L_w(1)))$$

求最小值

$$\alpha P_{wc}(d') \approx \theta(E(L_w(d')) - E(L_w(1)))$$

对不一致性的补偿

- **补偿函数** φ
  - $-\varphi = 0$，**用户对数据的不一致不敏感**

$$v(d) = wbr_b(P_{wc}(d) + r_l\left(E\left(L_w(d)\right) - E\left(L_w(1)\right)\right))$$

  - $-\varphi = r_c b$，**服务提供商按照成本的一定比率进行补偿**

$$v(d) = wb((r_b + \mathrm{r_c})P_{wc}(d) + r_l r_b\left(E\left(L_w(d)\right) - E\left(L_w(1)\right)\right))$$

  - $-\varphi = C$ ，**服务提供商花费一定管理成本处理不一致性**

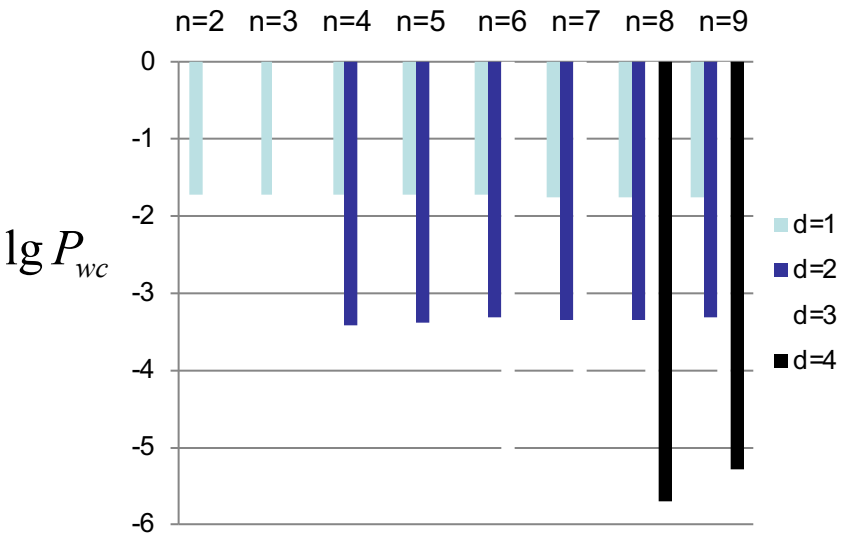
$$v(d) = w((br_b + \mathrm{C})P_{wc}(d) + r_l br_b\left(E\left(L_w(d)\right) - E\left(L_w(1)\right)\right))$$

w: 单位时间写请求个数，*b*:写请求成功时提供商的平均成本(比如，商品的价格等)，
r_b: 每个写请求的平均收益率，r_l:单位延迟增加导致写请求个数减少的比率。

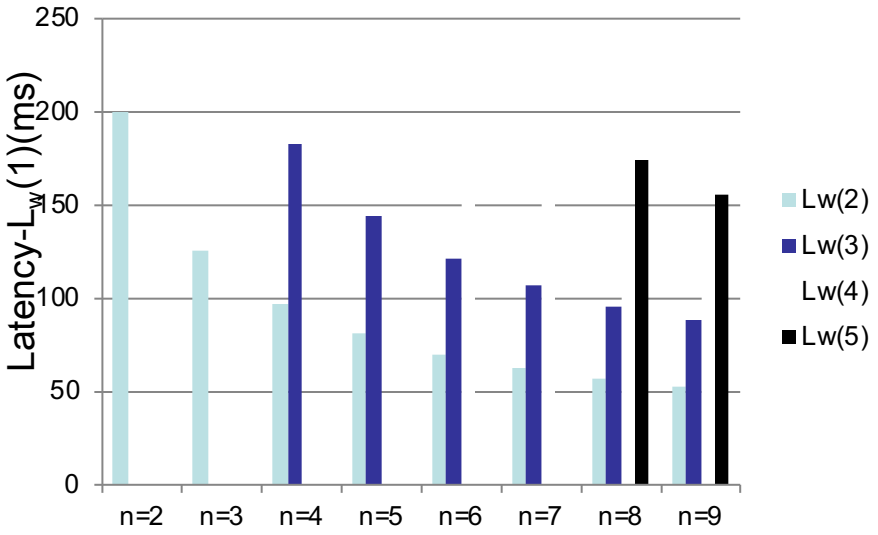
写冲突概率和延迟结果的验证

- 对每一千万个写请求，观察值：
  - 记录写请求丢失的数目N_{wl} 和写重复出现的次数 N_{wd}
  - P_{wc} 观察值为$(N_{wl} + N_{wd})/10,000,000$
- 写冲突概率
  - 任意n∈[2,9] 和d∈[1,[n/2]]
  - RMSE= 0.0009%, std.dev.=0.0052%
- 延迟
  - 任意n∈[2,9] 和d∈[1,n]
  - RMSE=0.013ms, std.dev.=0.019ms

d对一致性和延迟的影响

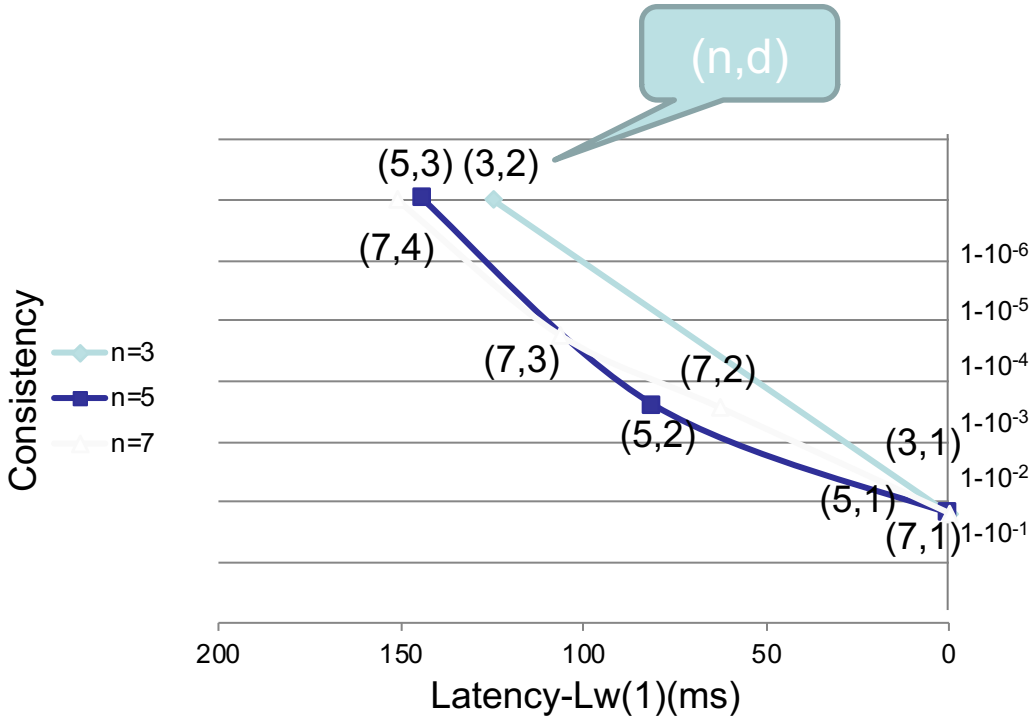


Impact of *d* on Consistency



Impact of *d* on Latency

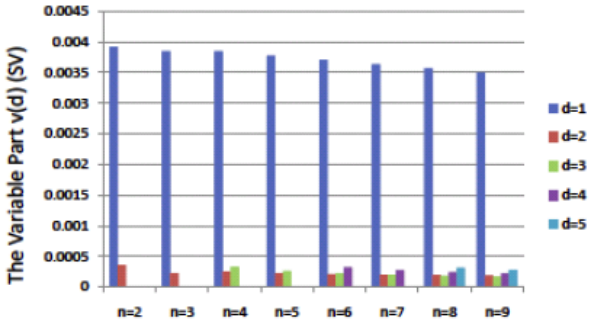
一致性 vs. 延迟


(n,d)
(5,3) (3,2)
(7,4)
(7,3)
(7,2)
(5,2)
(3,1)
(5,1)
(7,1)
1-10-6
1-10-5
1-10-4
1-10-3
1-10-2
1-10-1
Consistency
n=3
n=5
n=7
200
150
100
50
0
Latency-Lw(1)(ms)

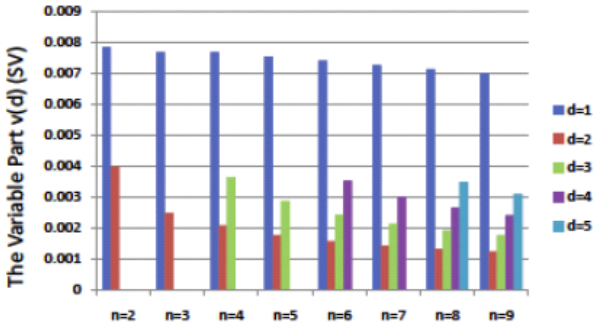
案例：在线购物服务

- **每个小时平均销售*S*个商品, 平均价格为*V***
  - **每个正常(一致)的购买会产生商品价格r_b比率的利润**
  - **每次异常(不一致)的购买会导致损失φ(即对订单冲突的补偿处理带来的成本)**
  - **服务对延迟的敏感程度**
    - Microsoft Bing： $r_l=0.0009\%$
    - Amazon.com： $r_l=0.01\%$
    - Google Search：$r_l=0.05\%$
  - **分别计算不同类型补偿函数下的*v(d)*值**

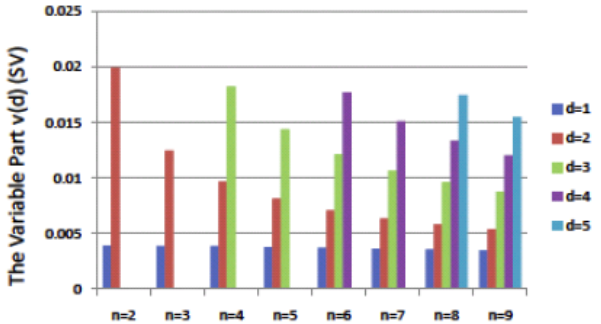
d值对系统效益的影响



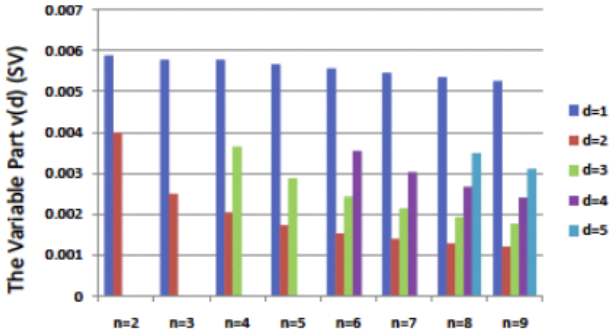
(a) $\varphi = 0, r_l = 0.0009\%$



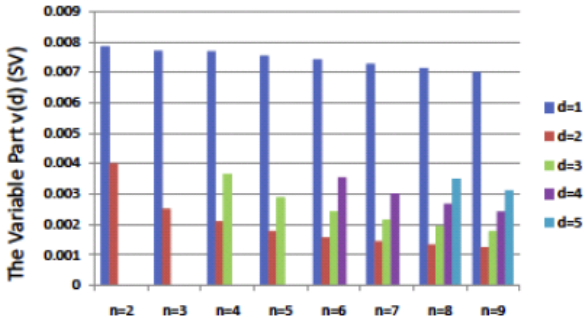
(b) $\varphi = 0, r_l = 0.01\%$



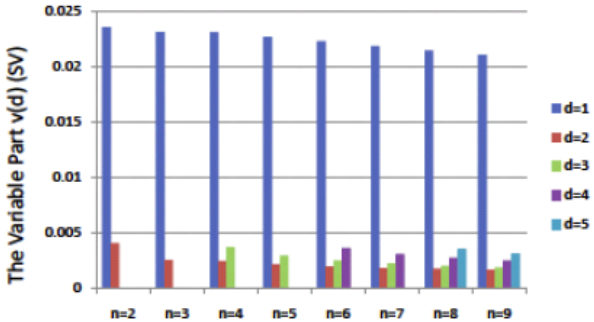
(c) $\varphi = 0, r_l = 0.05\%$



(d) $\varphi = r_c b, r_c = 0.5 r_b$ or $\varphi = C, C = 0.5 b r_b$



(e) $\varphi = r_c b, r_c = r_b$ or $\varphi = C, C = b r_b$

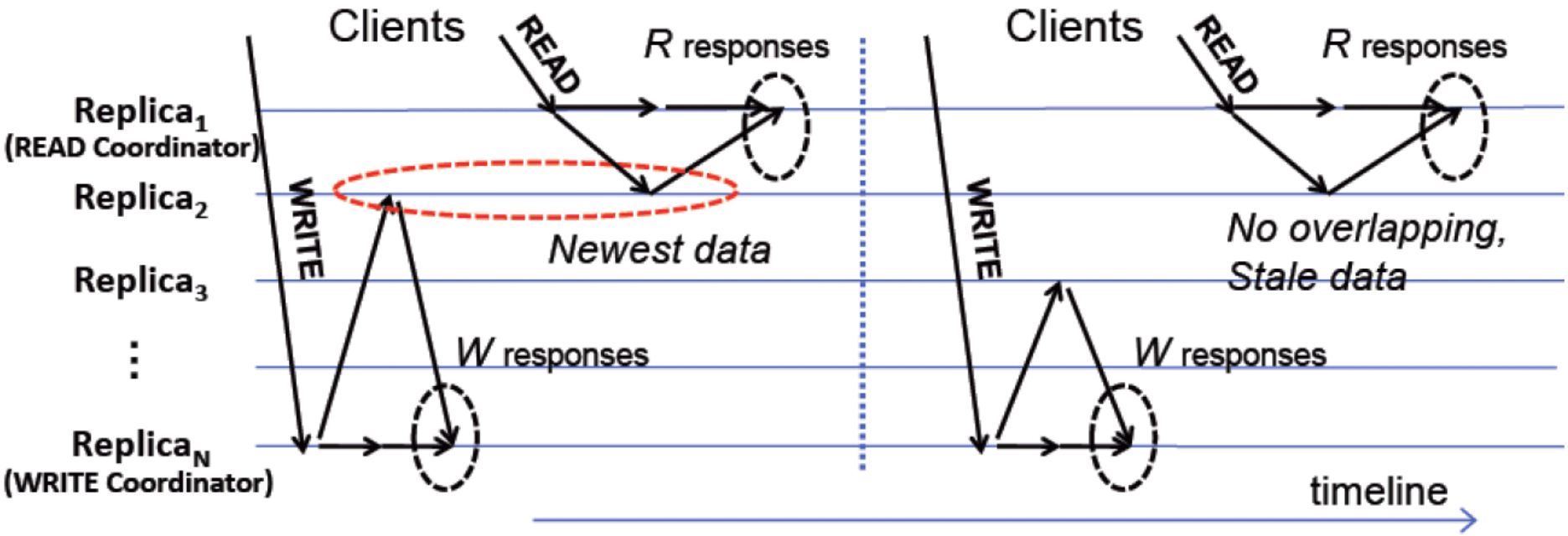


(f) $\varphi = r_c b, r_c = 5 r_b$ or $\varphi = C, C = 5 b r_b$

服务质量保障方法

- 四个方面
  - 一致性和性能的权衡
  - 一致性和可用性的权衡
  - 强一致性副本协议的性能优化
  - 高可用的最强弱一致性：因果一致性

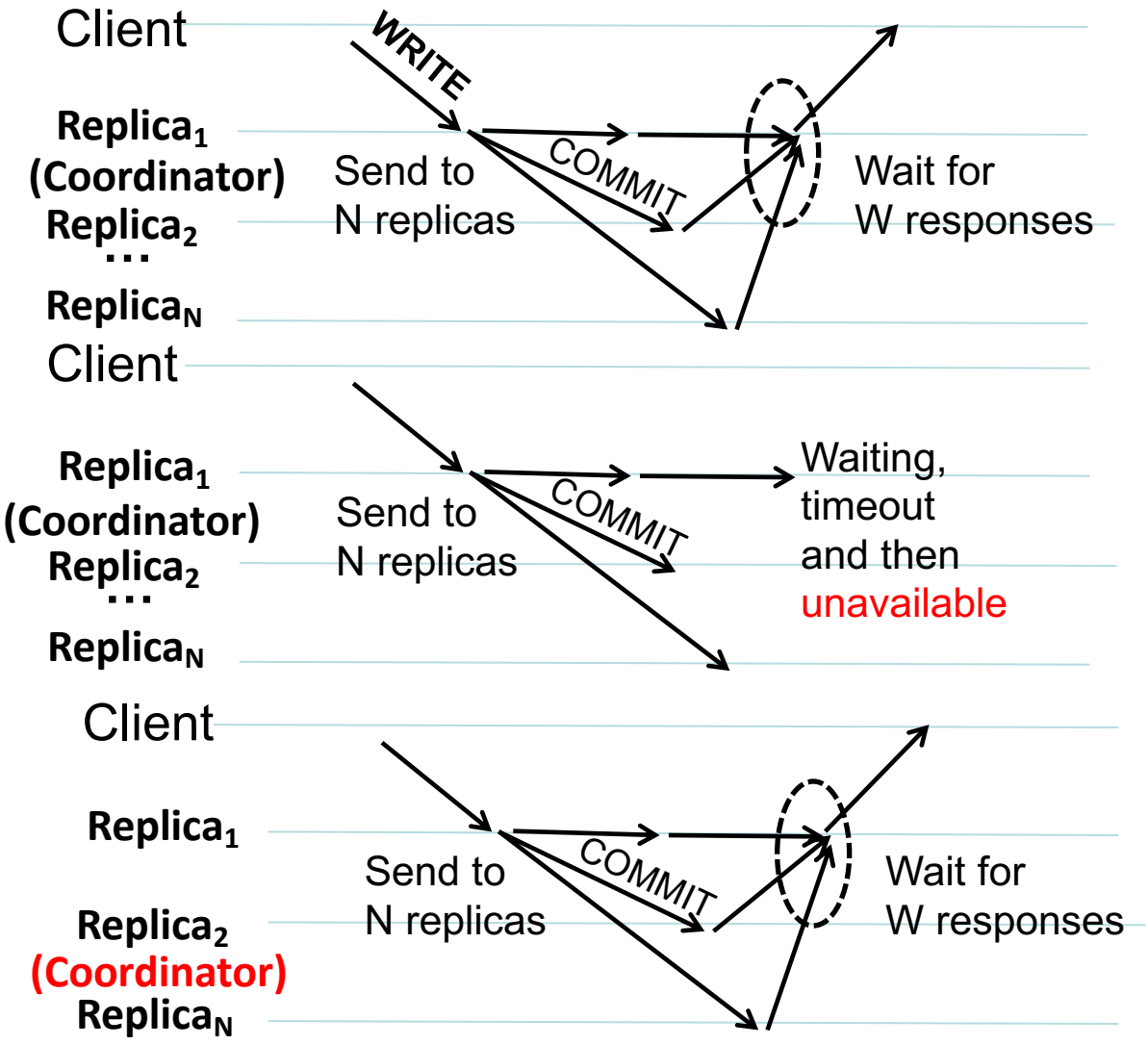
Quorum系统及其一致性

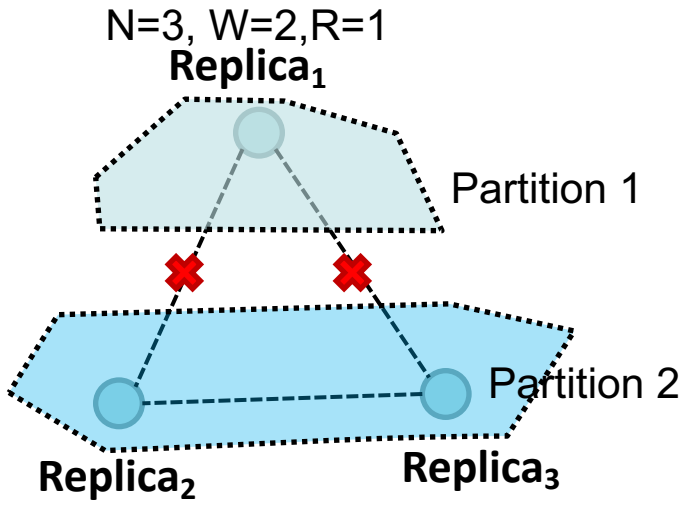


不一致性度量：读到旧数据的概率

$$p_{rs} = \binom{N-W}{R} \Big/ \binom{N}{R}$$

网络分割导致不可用

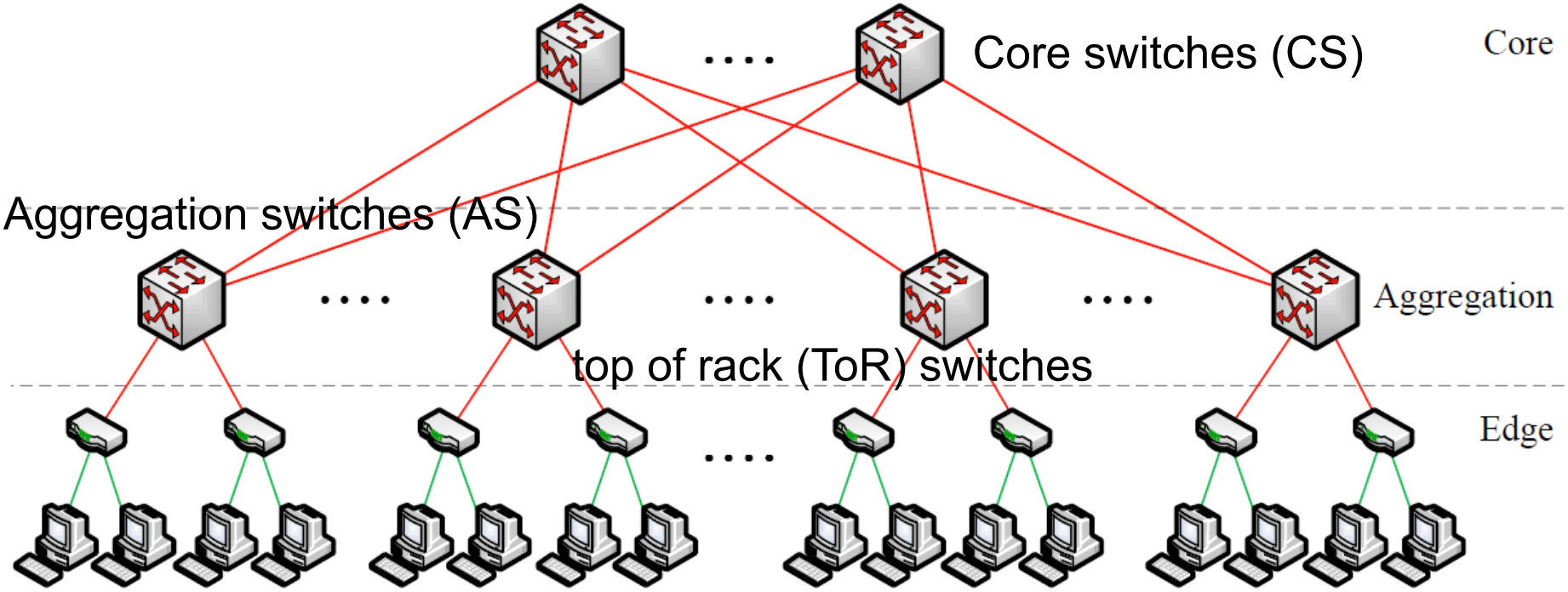




W/R 减小时，可用性上升

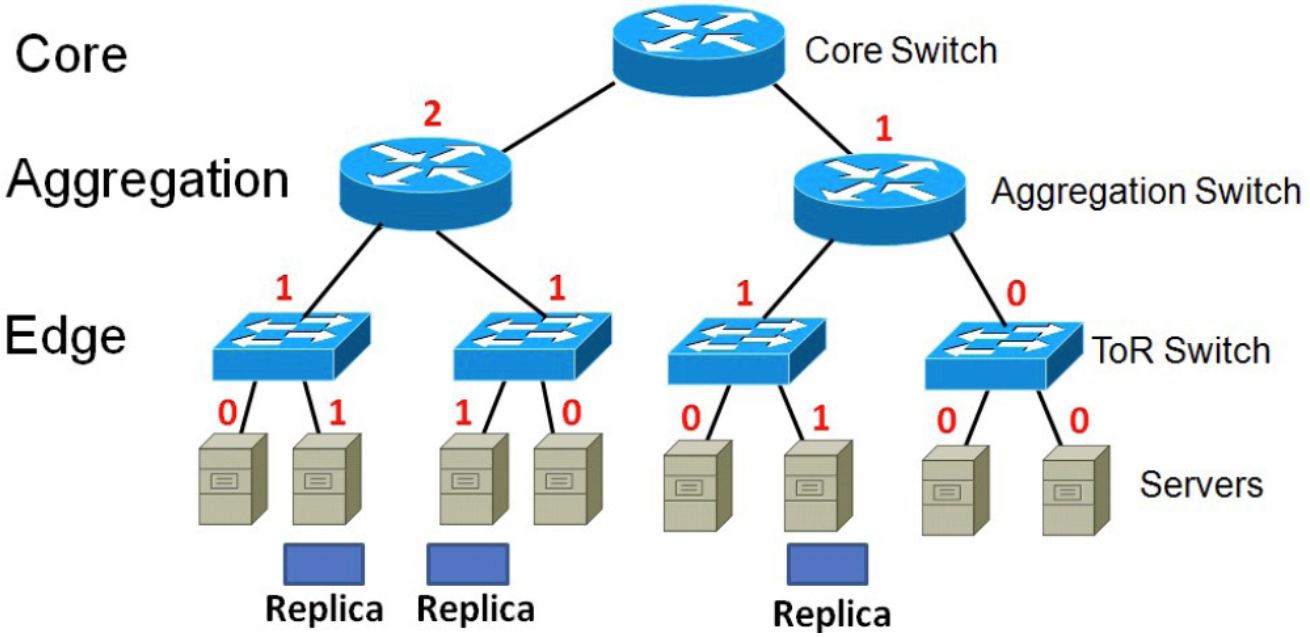
可用性的量化

- 典型数据中心网络中的Quorum系统可用性
  - 请求可到达活着的副本节点并完成Quorum系统的概率
  - 考虑二/三层基本树(2/3-tier basic tree), 胖树(fat tree), folded clos网络以及副本放置



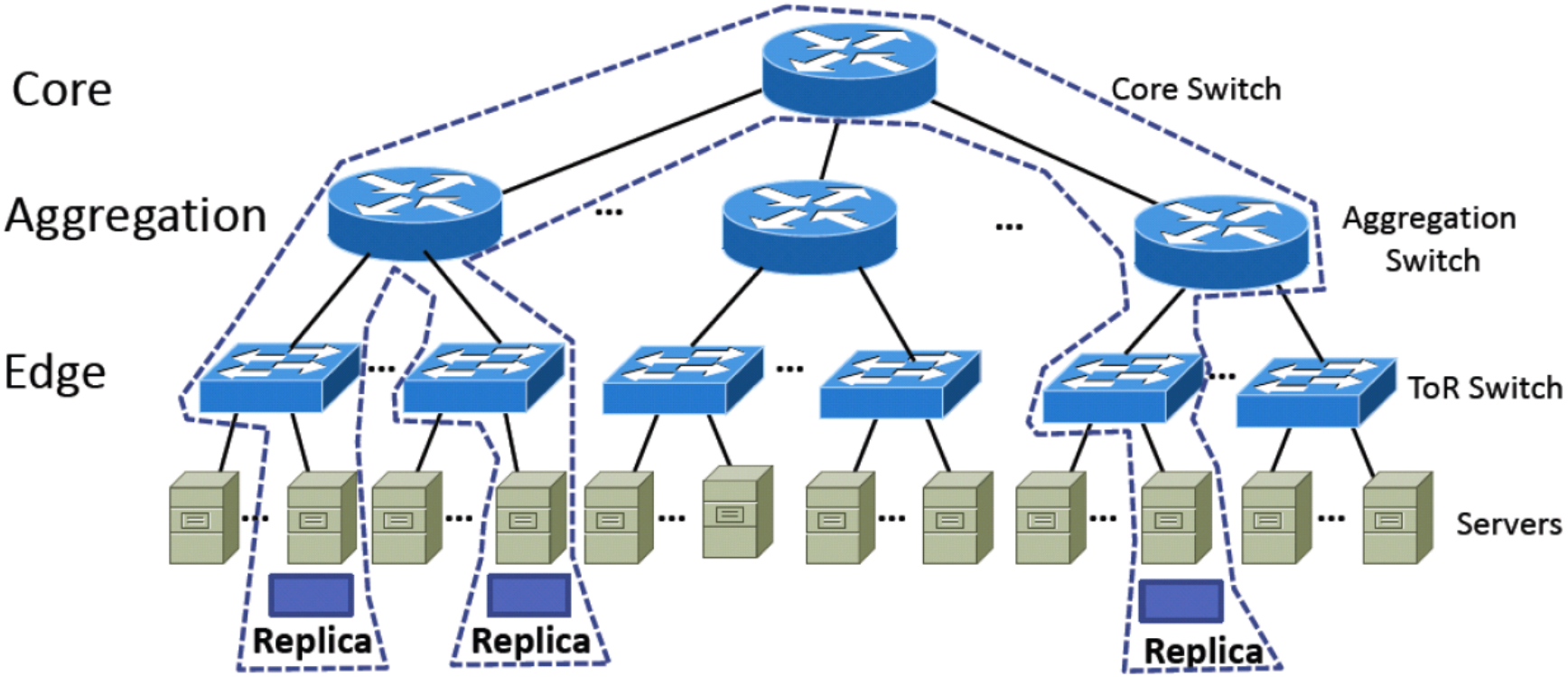
系统模型

- **Quorum系统模型QS(DCN,PM,W/R)**
  - ***DCN*∈{2层基本树(bt2)，3层基本树(bt3)，K胖树(ft)，folded clos网络(fc)}**
  - ***PM*: 副本放置在服务器上的0/1位置向量(机架，PM^{tor})**
  - ***W/R* :写Quorum和读Quorum的大小**



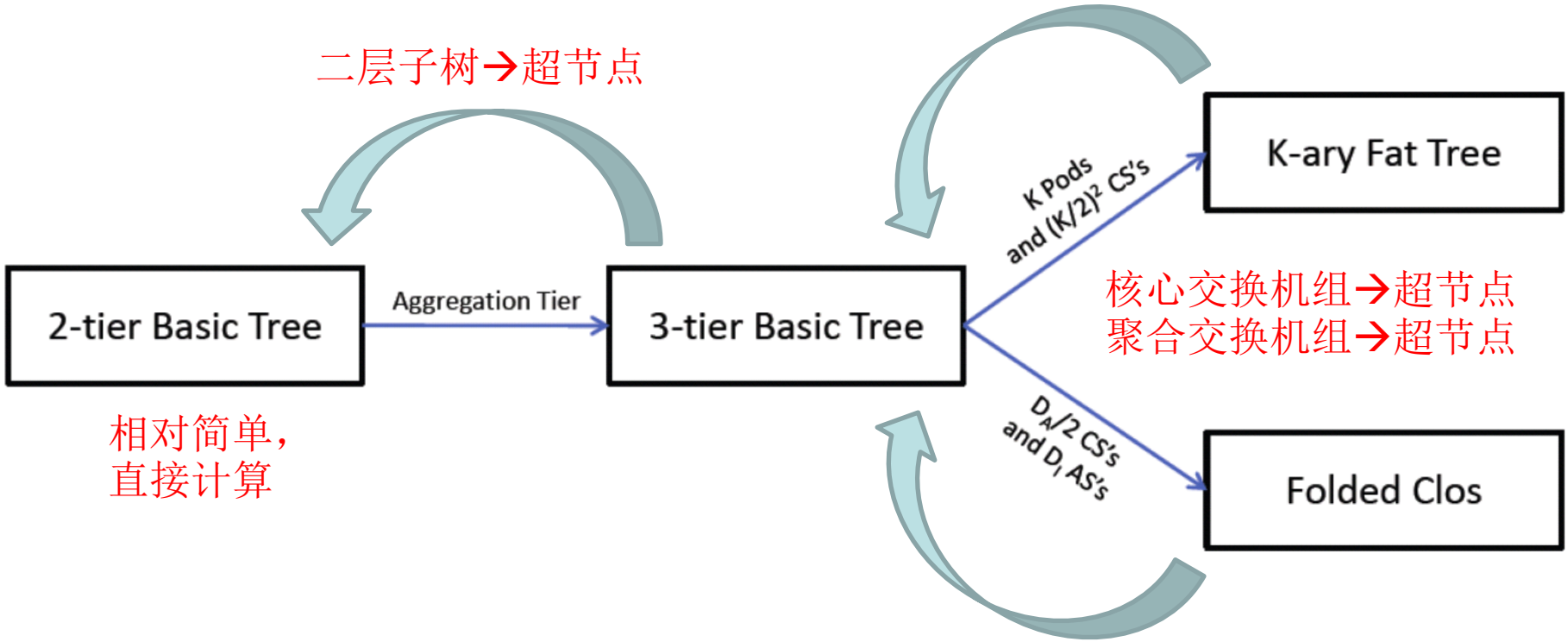
失效概率
(1) 核心交换机: P_c
(2) 聚合交换机: P_a
(3) ToR交换机: P_t
(4) 服务器: P_s
所有失效事件之间相互独立

可用性量化的思路(1)

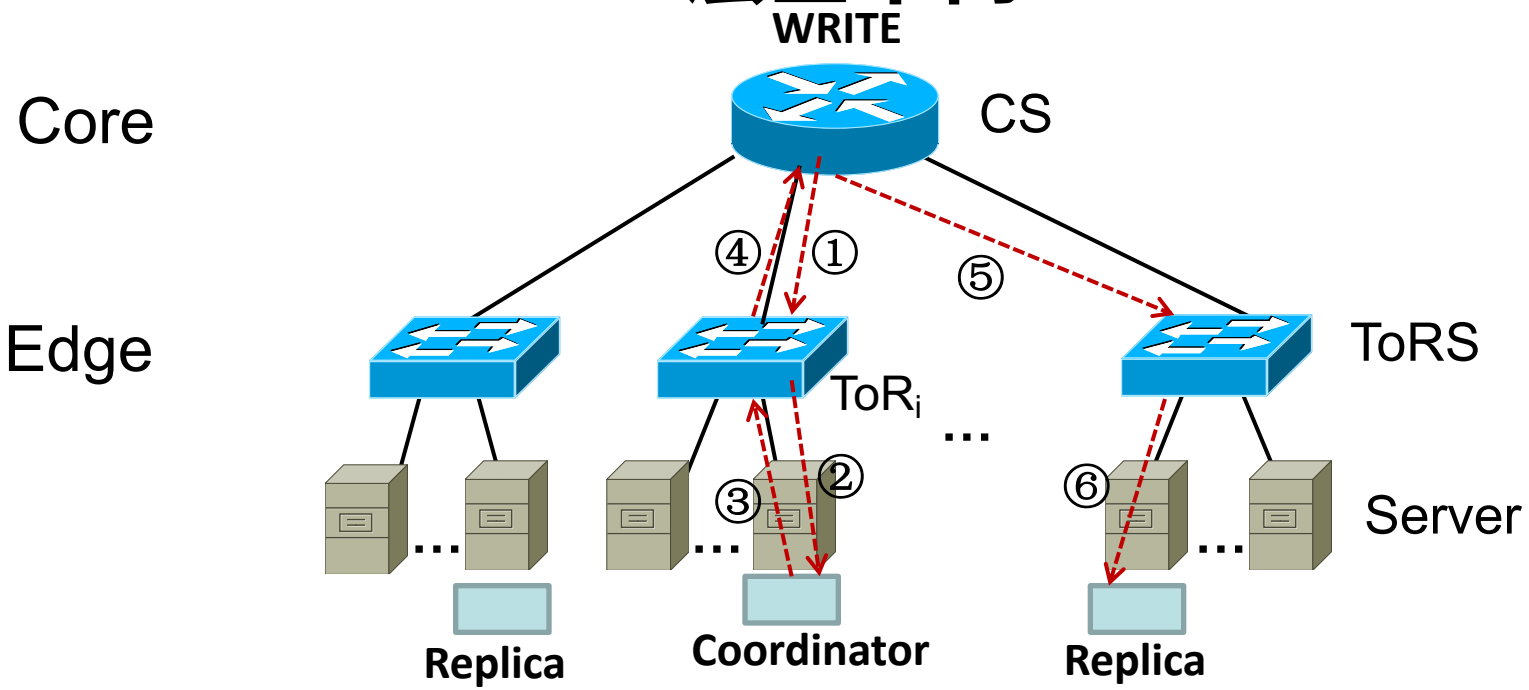


量化分析相关的子网络

可用性量化的思路(2)


二层子树→超节点
2-tier Basic Tree
Aggregation Tier
3-tier Basic Tree
K Pods and (K/2)² CS's
K-ary Fat Tree
核心交换机组→超节点
聚合交换机组→超节点
D_A/2 CS's and D_I AS's
Folded Clos
相对简单，
直接计算

二层基本树



A_i : 从活着的核心交换机开始，ToR_i 下可以访问到的正常工作的副本数目

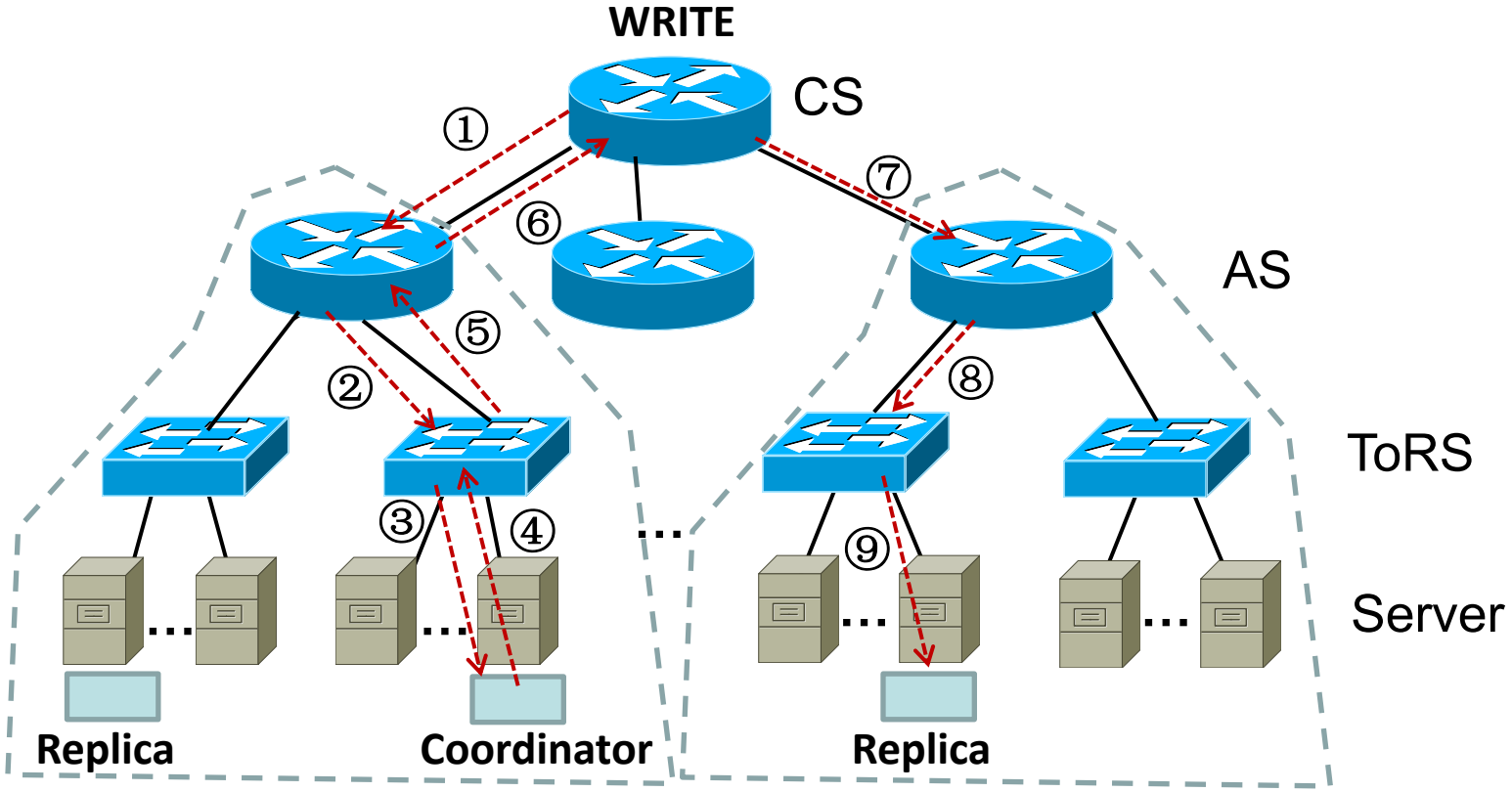
定义函数:

$$Q_{bt-2-tor}(x) = \sum_{m_1+m_2+\ldots+m_N=x} \prod_i P(A_i = m_i)$$

可用性：从外部可访问到的活着的副本数目不少于W

$$Avail(QS(bt2, PM, W)) = (1-P_s)\sum_{x=W}^{N} Q_{bt-2-tor}(x)$$

三层基本树

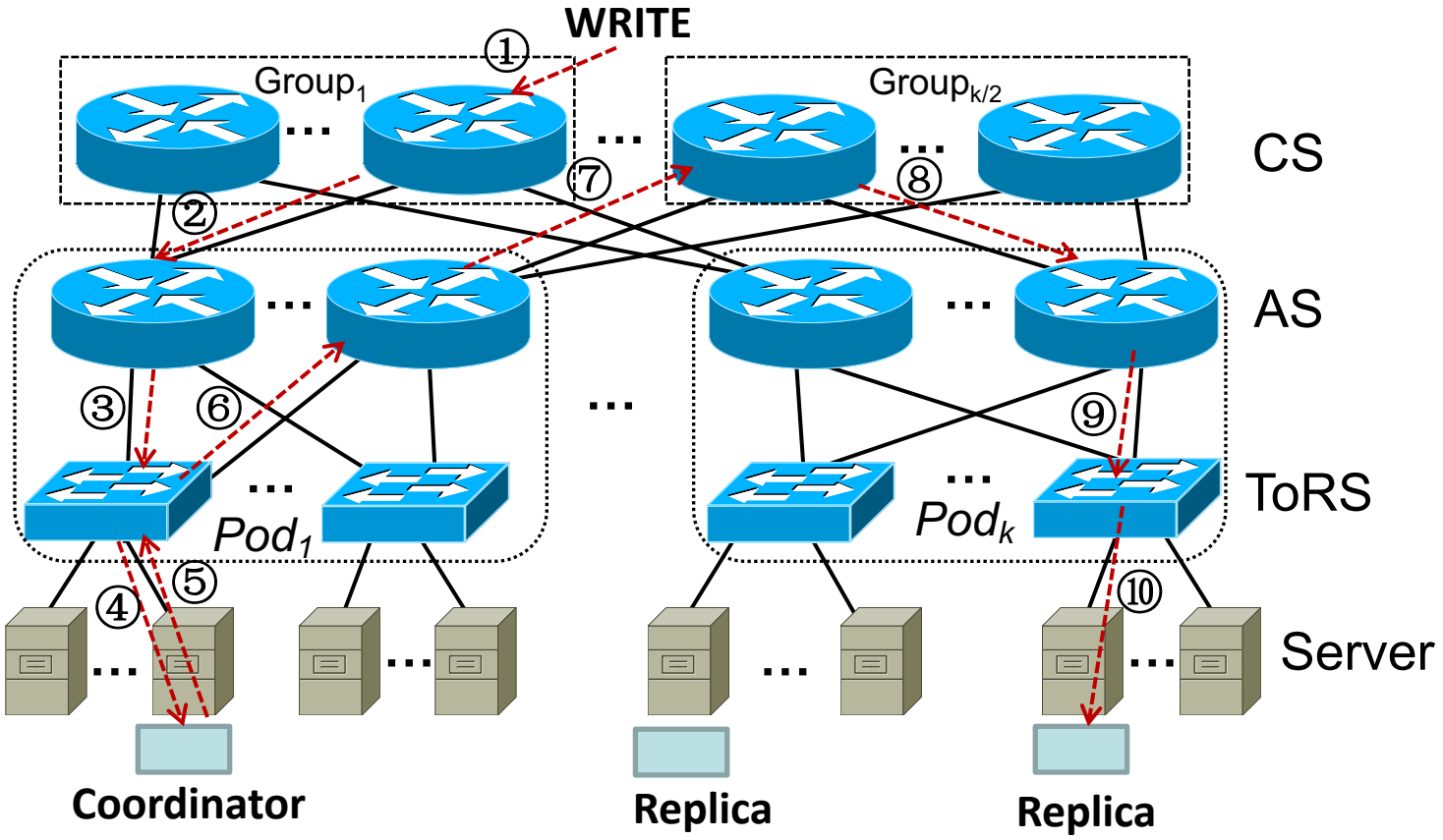


二层基本树　　重用 $Q_{bt-2-tor}(x)$

$X_{bt3\text{-}as}$: 从活着的核心交换机可访问到的正常副本数目

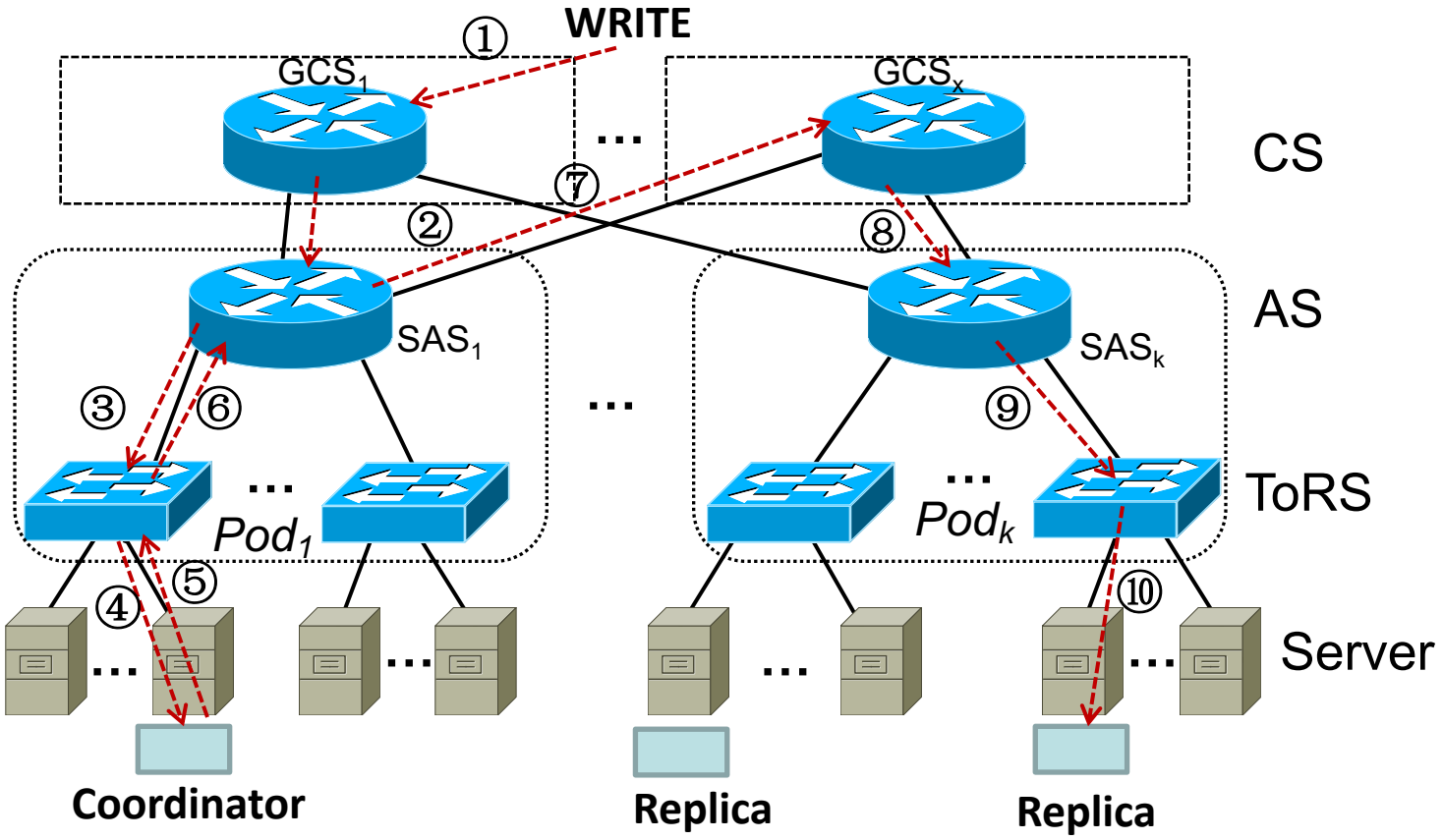
$$Avail(QS(bt3, PM, W)) = (1-P_s)\sum_{x'=W}^{N} \Pr(X_{bt3-as} = x')$$

K胖树



每k/2个核心交换机(CS)构成一组→超节点GCS
每个pod中的所有聚合交换机(AS)→超节点SAS

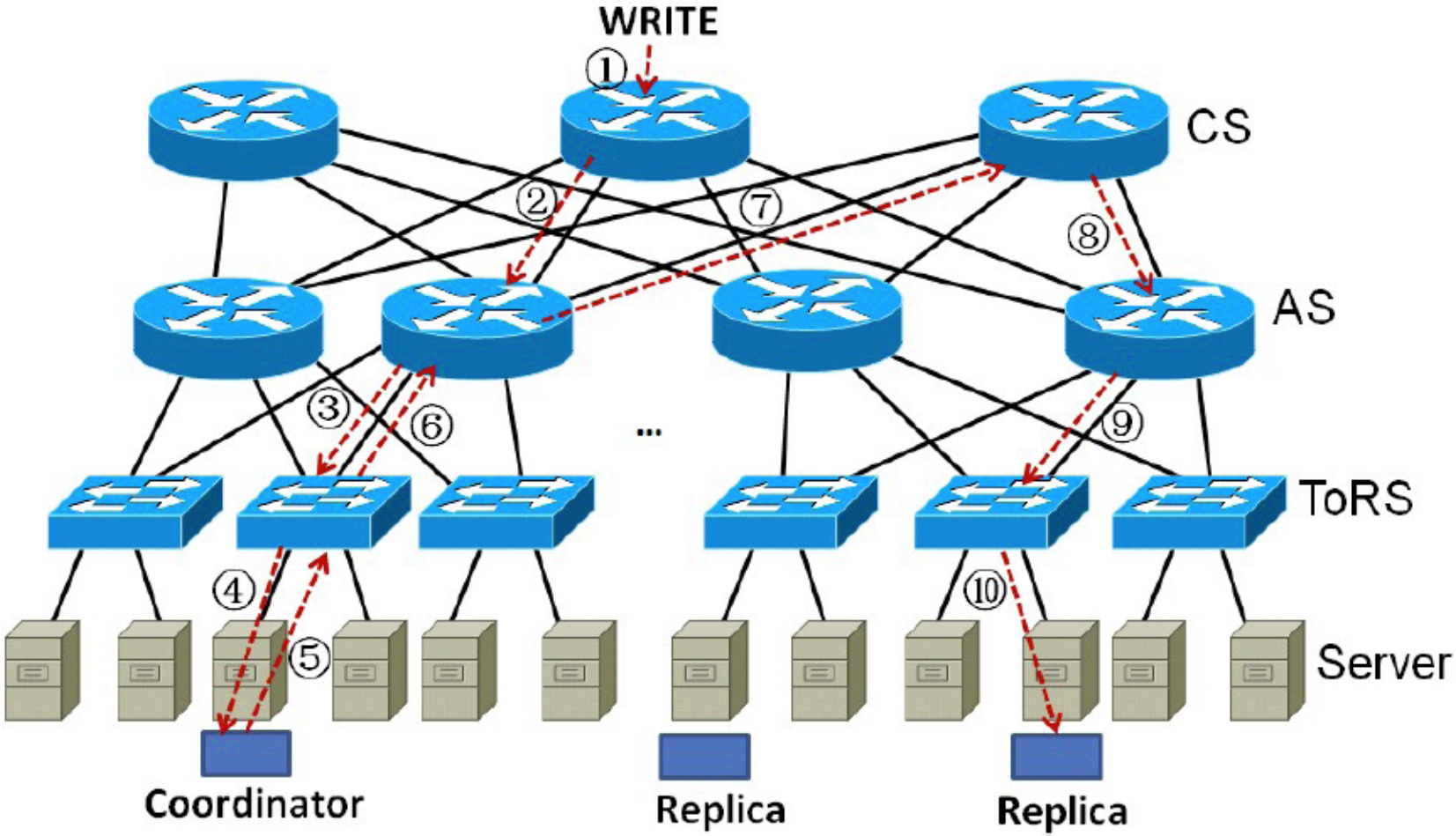
简化后的K胖树



假设恰好有x个GCS节点存活时，可重用3层基本树的结论，对条件概率求和可得到：

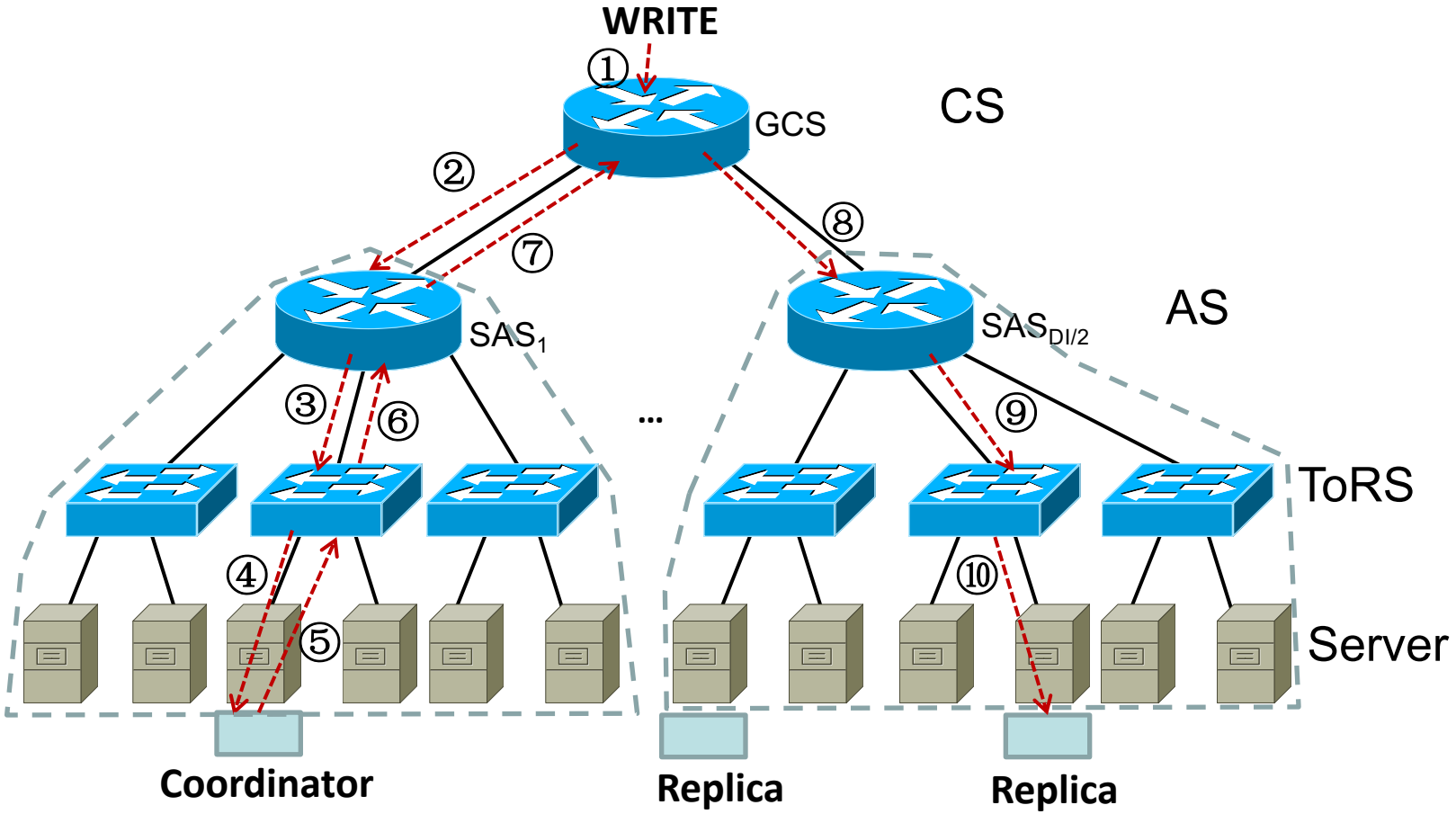
$$Avail(QS(ft,PM,W)) = \sum_{x'=W}^{N} \Pr(X_{ft-sas} = x')$$

Folded Clos网络



所有核心交换机(CS)构成一组→超节点GCS
每两个聚合交换机(AS)→超节点SAS

简化后的Folded Clos



可重用3层基本树的结论得到：

$$Avail(QS(fc,PM,W)) = (1-P_{gc}') \sum_{x'=W}^{N} \Pr(X_{fc-sas} = x')$$

推广

- 跨地域的多数据中心
  - 每个数据中心的失效事件独立，且任意副本的可用性可计算
  - 整体可用性等于所有可能副本放置的各个数据中心可用性的乘积之和
- 副本的最优放置问题

$$\text{Maximize} \quad Avail\left(QS\left(DCN, PM^{tor} = \langle r_1^t, r_2^t, \ldots, r_{n_{tor}}^t \rangle, W\right)\right)$$

Subject to

$$\begin{cases} \sum_{i=1}^{n_{tor}} r_i^t = N \\ 0 \le r_i^t, \quad r_i^t \in \mathbb{Z}\ (1 \le i \le n_{tor}) \end{cases}$$

案例分析

- **二层基本树(服务器不超过8K)，副本数*N*=3**
- **PM_1^{tor}=⟨1,1,1⟩，PM_2^{tor}=⟨2,1,0⟩和 PM_2^{tor}=⟨3,0,0⟩**

W	一种Cassandra配置	常见的Hadoop配置
1	$Avail(PM_1^{tor}) > Avail(PM_2^{tor}) > Avail(PM_3^{tor})$	
2	$Avail(PM_1^{tor}) \leq Avail(PM_2^{tor}) < Avail(PM_3^{tor})$	$1 + 2P_sP_t \leq 2P_t + 2P_s$
	$Avail(PM_2^{tor}) < Avail(PM_1^{tor}) \leq Avail(PM_3^{tor})$	$2P_t + 2P_s < 1 + 2P_sP_t \leq 2P_t + 4P_s$
	$Avail(PM_2^{tor}) < Avail(PM_3^{tor}) < Avail(PM_1^{tor})$	$1 + 2P_sP_t > 2P_t + 4P_s$
3	$Avail(PM_1^{tor}) < Avail(PM_2^{tor}) < Avail(PM_3^{tor})$	

一致性 vs. 可用性

- **写请求比率为α，** 系统可用性：

$$Avail(W,R) = max_{PM}(\alpha Avail(QS(W)) + (1-\alpha)Avail(QS(R)))$$

- 一致性，用读到最新数据的概率表示：

$$Consistency(W,R) = 1 - \binom{N-W}{R} / \binom{N}{R}$$

- 构建可用性-一致性(***W,R***)表格，查询规则：
  - **具有较小[N_n]值的较优** $N_n = -lg(1 - Avail(W,R))$
  - **相同 [N_n]值时，较大一致性的为优**
  - **相同 [N_n]值和一致性时，较小的α*W*+(1- α)*R*将较优**

可用性量化结果的验证

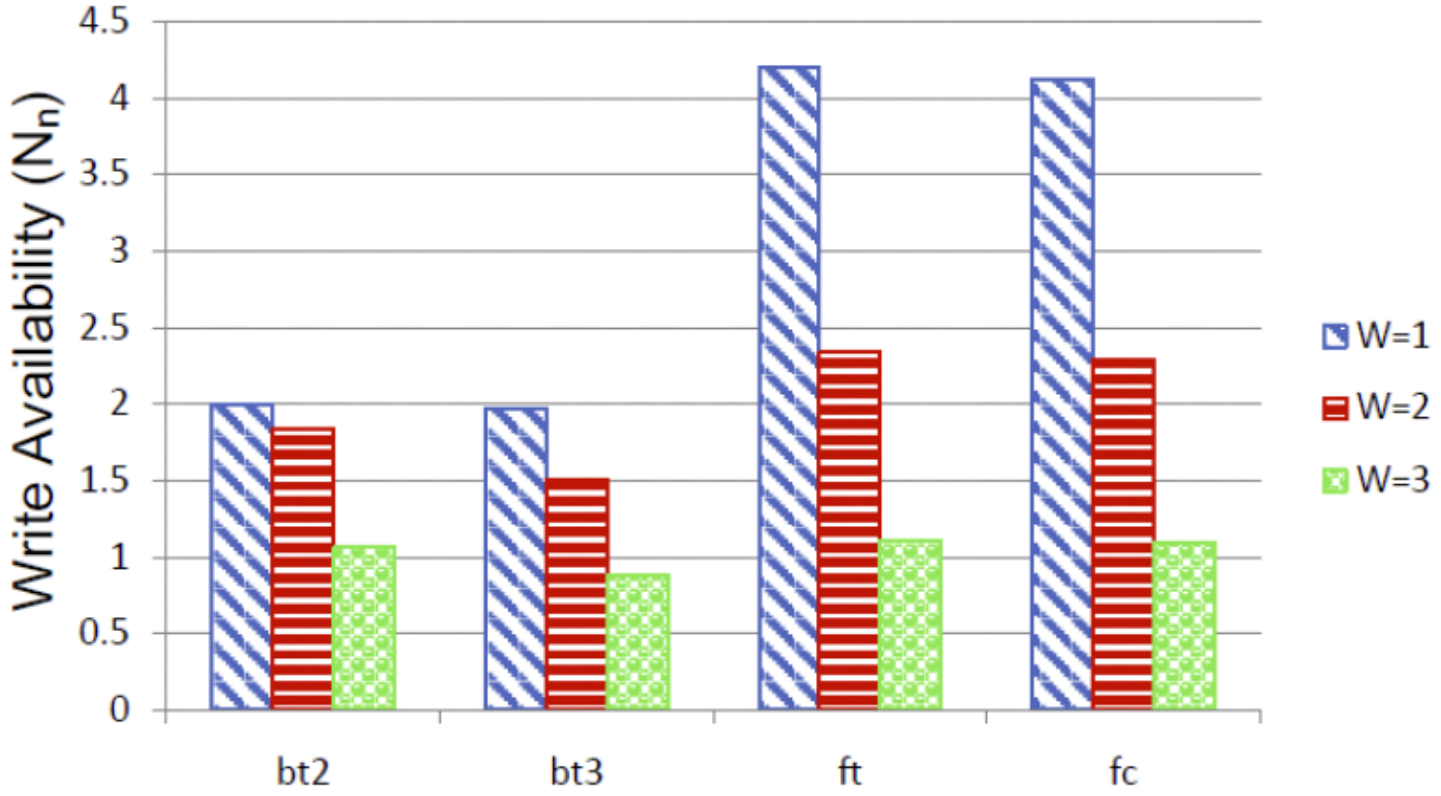
- 任意$N \in [3,9]$ 和$W \in [1,N]$, 以及所有可能副本放置

Table 3: Write Availability Validation

	RMSE	STD. DEV.
2-tier Basic Tree	0.000472%	0.000417%
3-tier Basic Tree	0.000763%	0.000668%
K-ary Fat Tree	0.00117%	0.000934%
Folded Clos	0.000845%	0.000689%

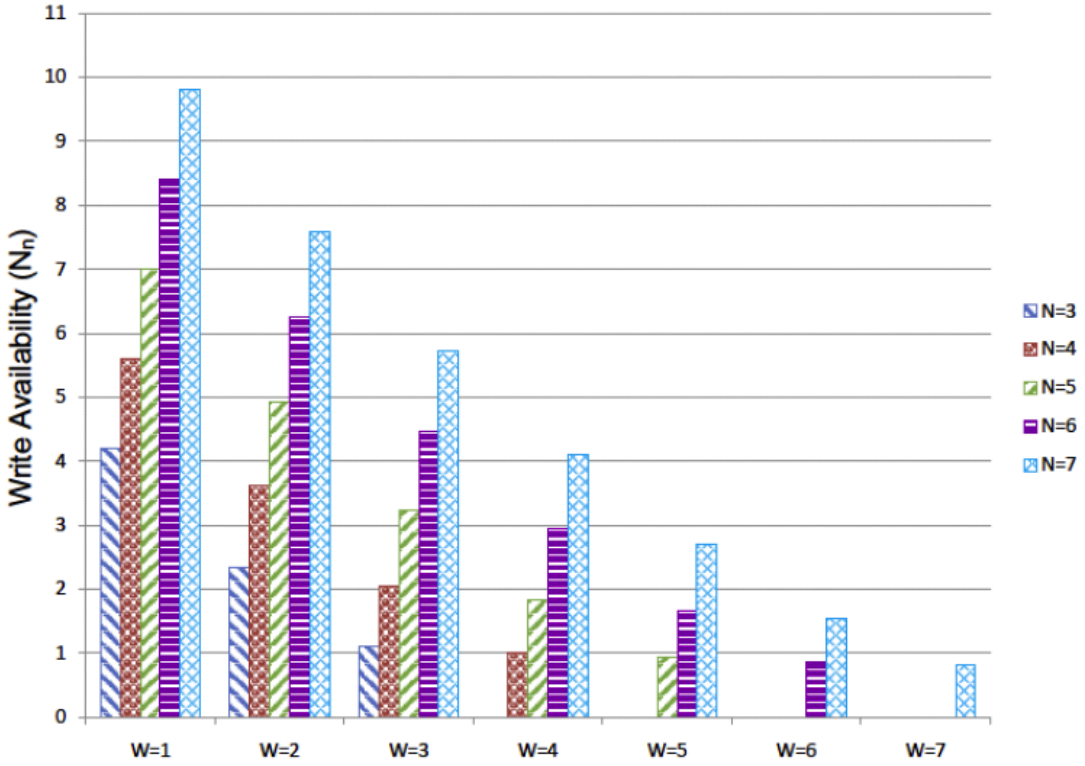
- 量化精度在4个9之上

*W*值对可用性的影响



- *N*=3, *W*的变化会改变可用性9的个数(ft, fc)

副本个数*N*对可用性的影响



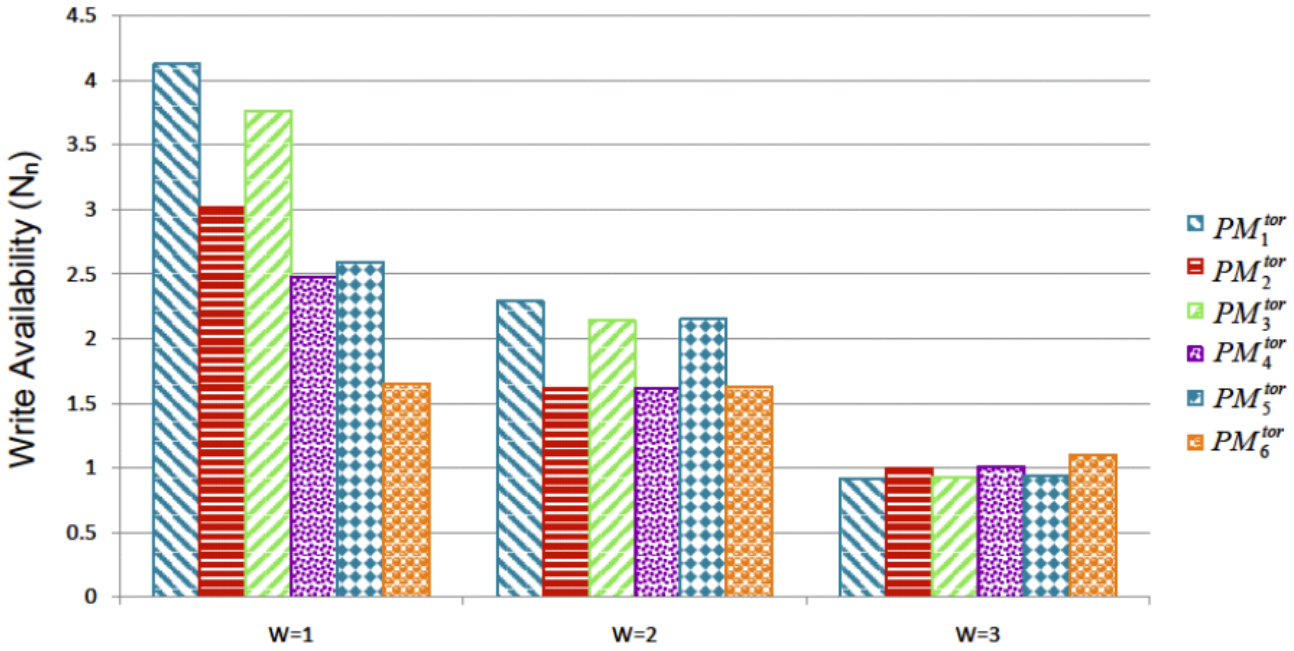
- 胖树，*N*的变化一定会改变可用性9的个数

副本放置*PM*对可用性的影响

- **Folded Clos网络, *N*=3，6种副本放置**

常见的Hadoop配置

- $PM_1^{tor} = \langle 1,0,0,1,0,0,1,0,0 \rangle$
- $PM_2^{tor} = \langle 2,0,0,1,0,0,0,0,0 \rangle$
- $PM_3^{tor} = \langle 1,1,0,1,0,0,0,0,0 \rangle$
- $PM_4^{tor} = \langle 2,1,0,0,0,0,0,0,0 \rangle$
- $PM_5^{tor} = \langle 1,1,1,0,0,0,0,0,0 \rangle$
- $PM_6^{tor} = \langle 3,0,0,0,0,0,0,0,0 \rangle$



可用性-一致性优化权衡

- **胖树，读主导的系统(α=0.05)**

Table 4: Availability-Consistency Table

$\langle A, C \rangle$	$R = 1$	$R = 2$	R=3
W=1	$\langle 4, 0.333 \rangle$	$\langle 2, 0.667 \rangle$	$\langle 1, 1.0 \rangle$
W=2	$\langle 3, 0.667 \rangle$	$\langle 2, 1.0 \rangle$	$\langle 1, 1.0 \rangle$
W=3	$\langle 2, 1.0 \rangle$	$\langle 2, 1.0 \rangle$	$\langle 1, 1.0 \rangle$

- 可用性需求
  - **99.9%→(W,R)=(2,1)最优**
  - **99%→(W,R)=(3,1)最优**

服务质量保障方法

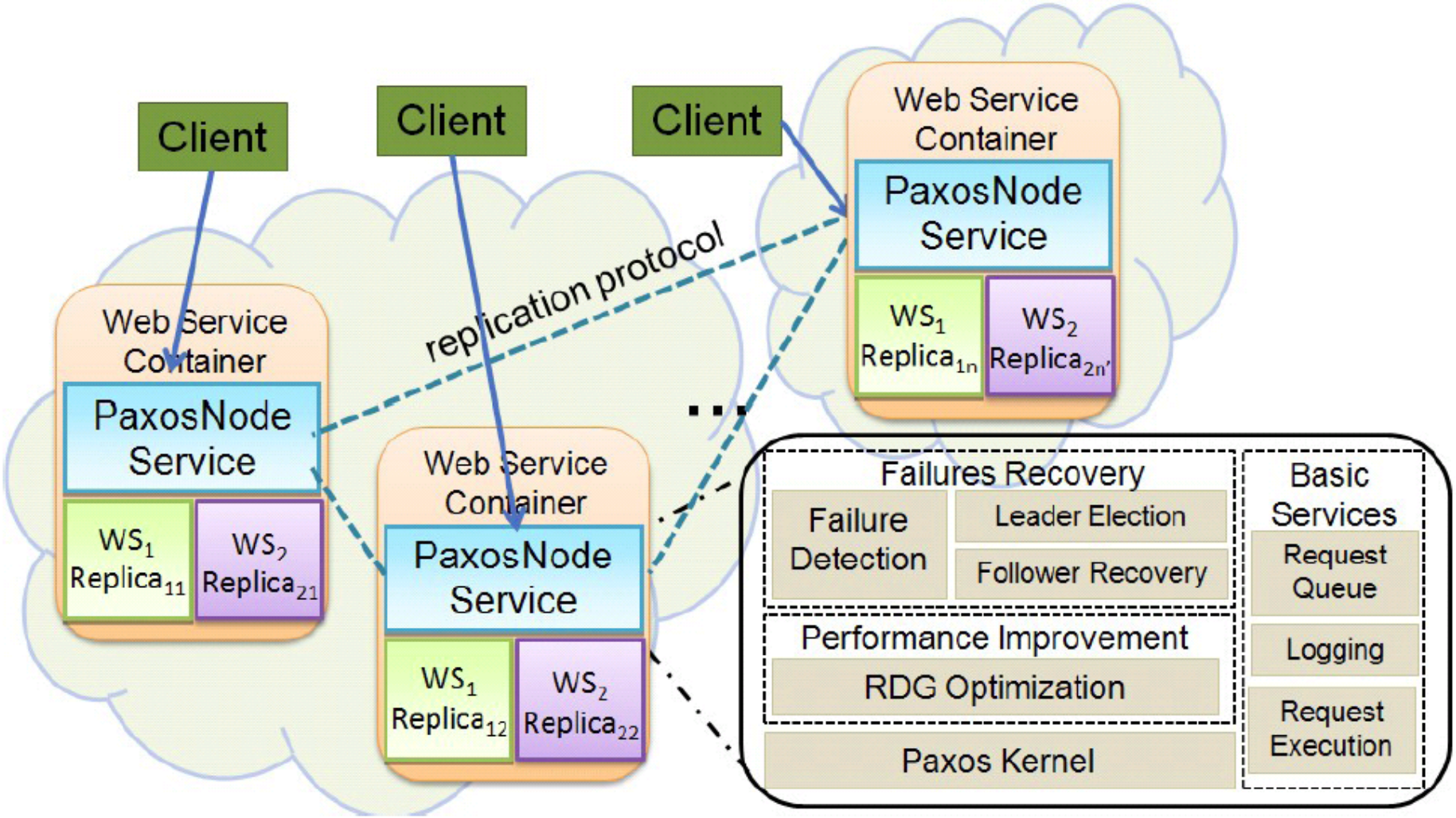
- 四个方面
  - 一致性和性能的权衡
  - 一致性和可用性的权衡
  - 强一致性副本协议的性能优化
  - 高可用的最强弱一致性：因果一致性

Web服务与强一致性

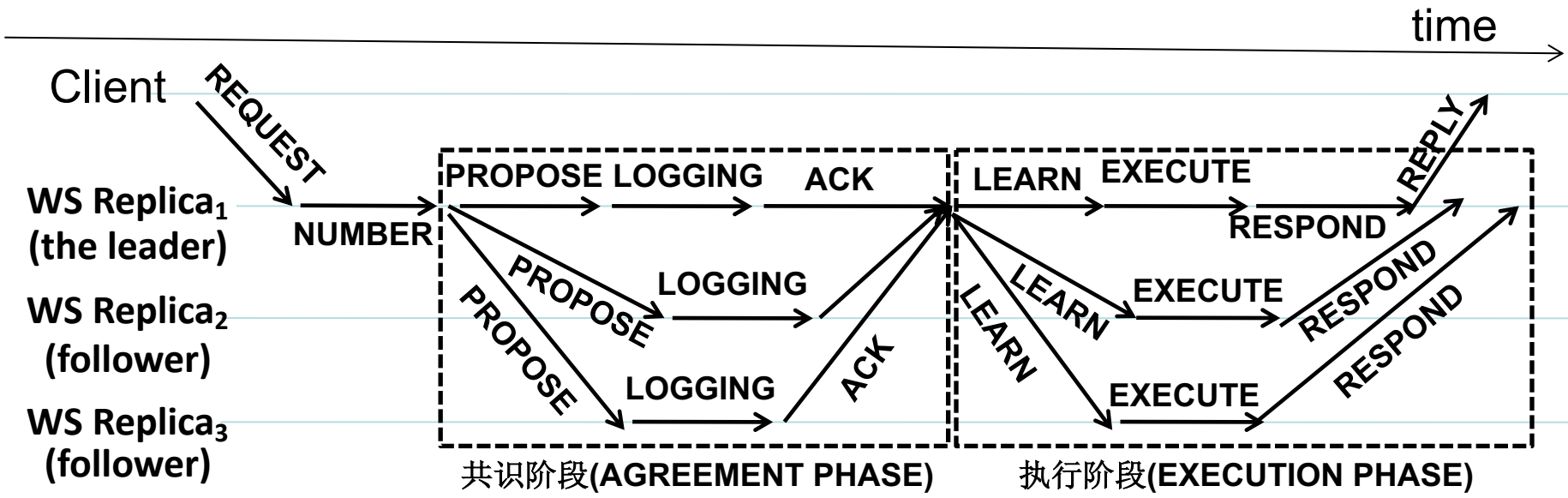
- **Web服务：接口明晰的HTTP层服务**
  - **Ebay Web服务**
  - **AWS**

- **金融等关键领域,要求强一致性**
  - **强一致性副本协议**
    - 主从备份
    - 两阶段提交协议(2PC)
    - 组通信
    - Paxos

副本框架Rep4WS


Client
Client
Client
Web Service Container
PaxosNode Service
WS1 Replica1n
WS2 Replica2n'
replication protocol
Web Service Container
PaxosNode Service
WS1 Replica11
WS2 Replica21
Web Service Container
PaxosNode Service
WS1 Replica12
WS2 Replica22
...
Failures Recovery
Failure Detection
Leader Election
Follower Recovery
Performance Improvement
RDG Optimization
Paxos Kernel
Basic Services
Request Queue
Logging
Request Execution

基于Paxos的副本协议



基于请求依赖图(RDG)的
并发执行优化

NUMBER: 对请求按顺序编号，1,2,3…

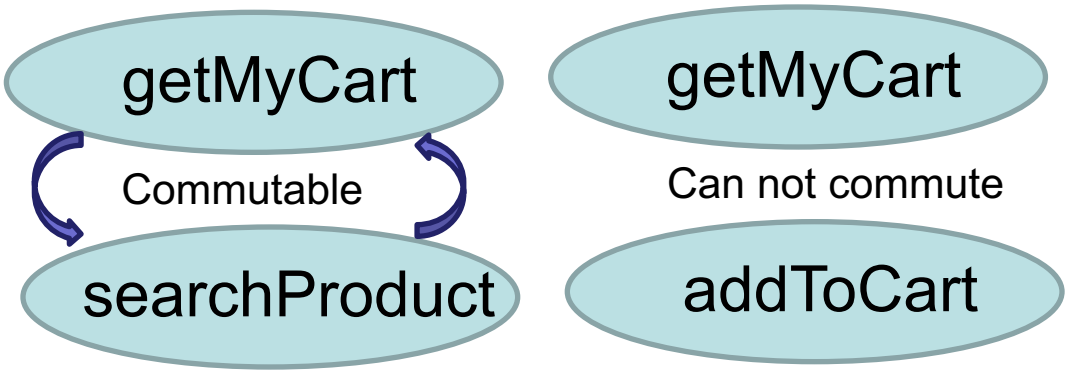
ACK: 主节点等待n/2+1个ACK消息

EXECUTE: 按照请求的编号按序执行，下一个编号请求未到来时则进行等待

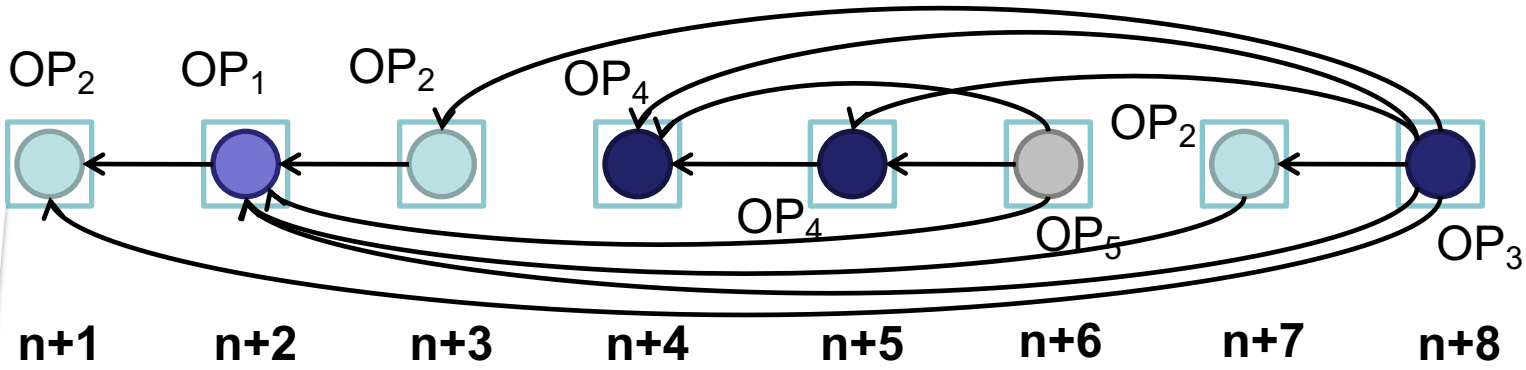
AGREEMENT PHASE: 以窗口大小α进行流水线并行(Pipeline Concurrency)

请求间的可交换性

- 软件服务通过API对外提供接口
  - 某些接口的请求之间交换执行顺序不会影响一致性，这些请求称为可交换的(commutable)
  - 以请求为点，以不可交换关系(也称为依赖关系)建立边，形成的图为请求依赖图(RDG)
- WebShopService
  - getMyCart
  - addToCart
  - searchProduct
  - orderProduct
  - modifyMyProfile



请求依赖图

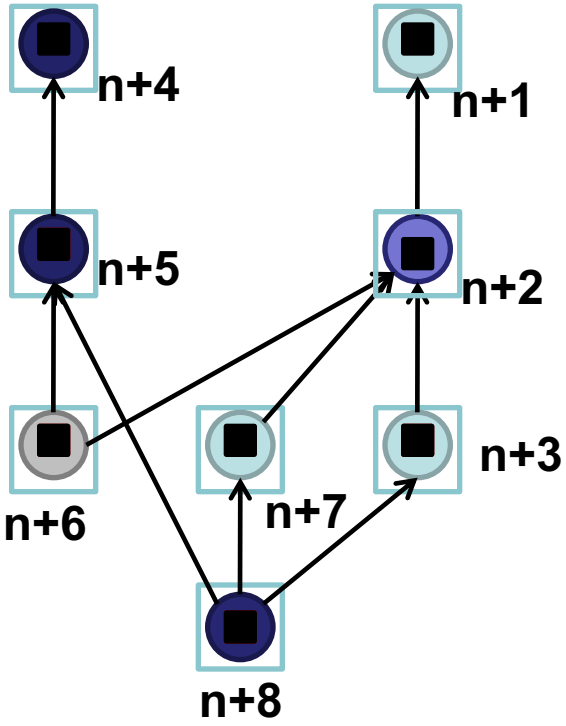


访问操作OP_2的请求n+1

WS					
OP_1	×	×	×	√	×
OP_2		√	×	√	√
OP_3			√	×	√
OP_4				×	×
OP_5					√

可交换

基于RDG的优化执行



RDG of Pipeline[n+1,n+8]

LEARN:

(n+1) ⇒ EXECUTE ⇒ Executed

(n+5) ⇒ Wait ⇒ EXECUTE ⇒ Executed

(n+3) ⇒ Wait ⇒ EXECUTE ⇒ Executed

(n+8) ⇒ Wait ⇒ EXECUTE ⇒ Executed

(n+4) ⇒ EXECUTE ⇒ Executed

(n+6) ⇒ Wait ⇒ EXECUTE ⇒ Executed

(n+2) ⇒ EXECUTE ⇒ Executed

(n+7) ⇒ EXECUTE ⇒ Executed

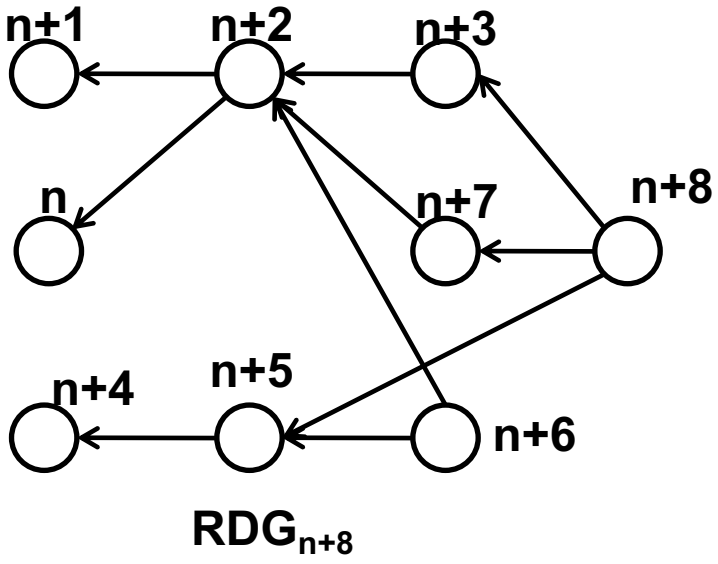
等待时间: 3+4+4+1=12 units

不使用RDG的等待时间：5+4+4+2+1=16 units

RDG的复杂度以及优化算法的正确性

- **RDG复杂性**

$$Complexity(RDG_{n+i}) = \frac{\sum_{j=0}^{i} C\left(DNS\left(r_{n+j}\right)\right)}{i+1}$$



RDG_{n+8}

(1)边数为0的RDG复杂度为0；
(2)请求数为(i+1)，任意两点间都有边的RDG复杂度最大,为(i/2)。

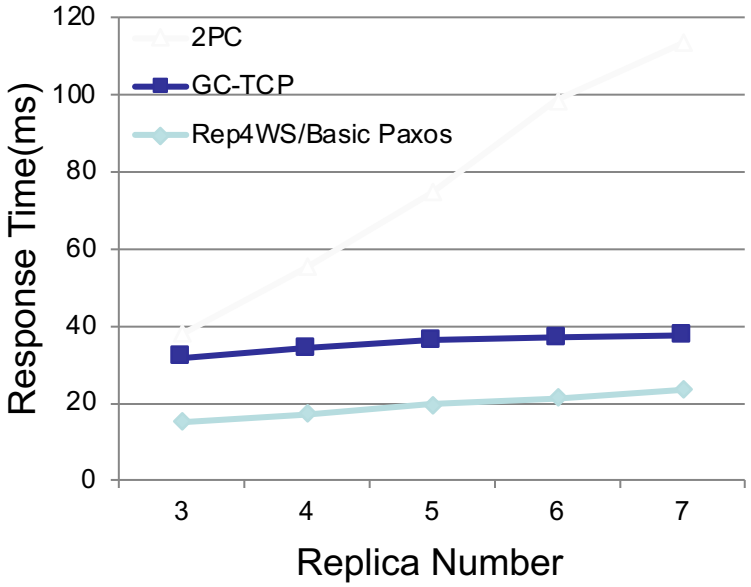
平均响应时间(延迟)与RDG复杂度成线性关系

- **优化算法的正确性**
  - **基于RDG的优化会改变请求执行顺序，但不会改变每个请求的输出结果和最终的状态**

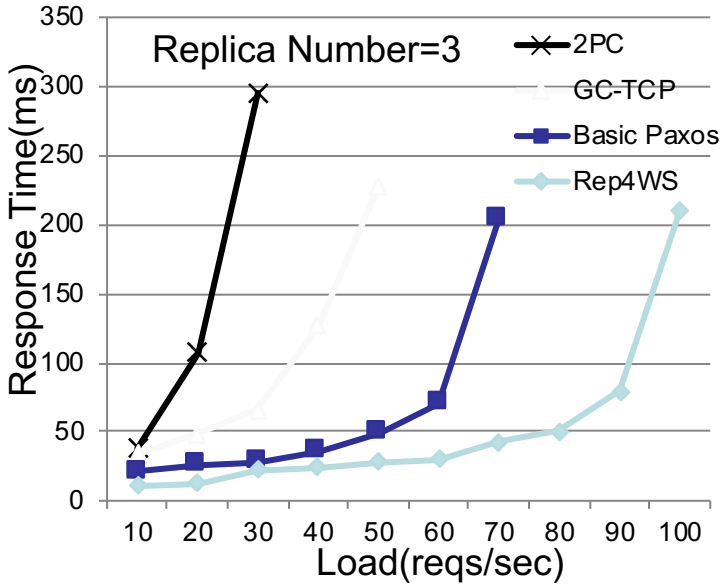
失效恢复

- 主节点(leader)失效
  - 基于Paxos的新主节点选举
  - 免回滚，状态同步
- 从节点(follower)失效
  - 本地恢复(checkpoint)
  - 如果中间leader发生变化，要移除与那时leader不一致的请求
  - 追赶到当前leader的状态
- 恢复→新的一致性状态

性能比较

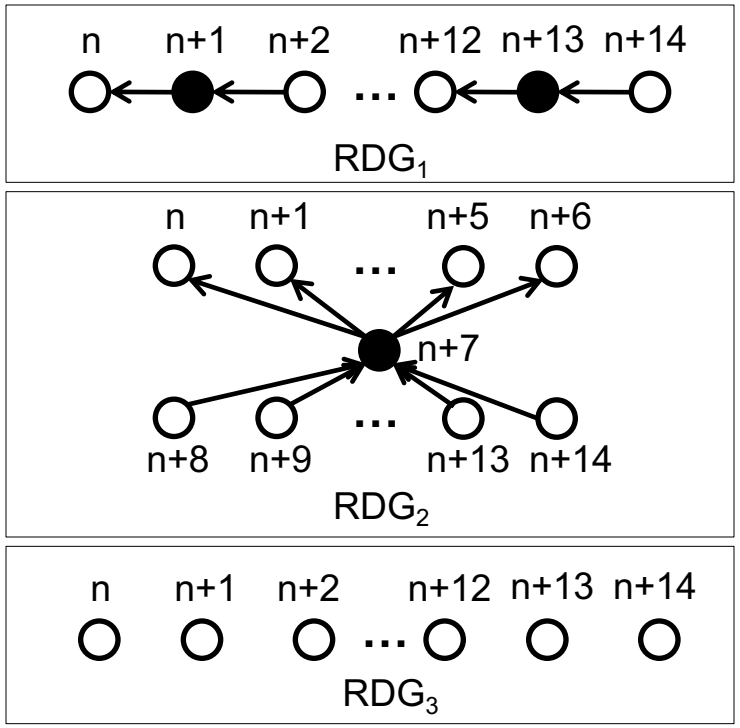


串行执行



并行执行

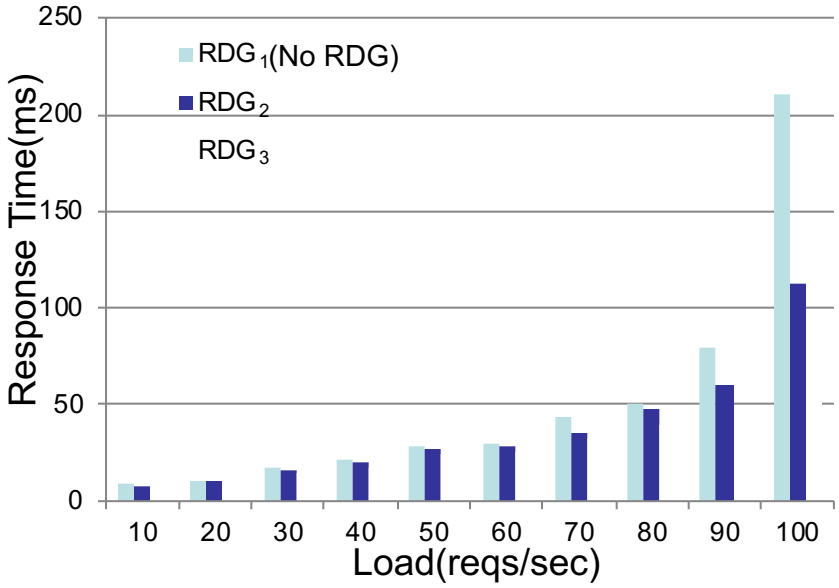
RDG优化的程度



read operation write operation

三种典型RDGs

复杂度依次为7，4.2，0



RDG3平均响应时间比RDG1低15%-66%，且平均要低26%。

服务质量保障方法

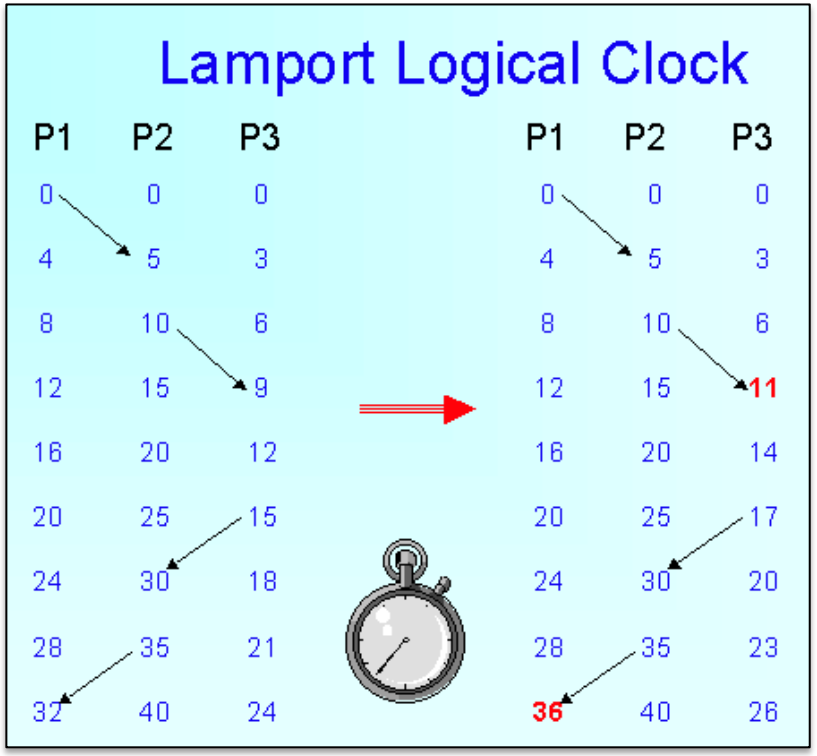
- 四个方面
  - 一致性和性能的权衡
  - 一致性和可用性的权衡
  - 强一致性副本协议的性能优化
  - 高可用的最强弱一致性：因果一致性

因果一致性

- 什么是操作间的**因果关系**？
  - 操作O的执行结果取决于操作P的执行结果，那么O和P之间存在着**因果关系**，记为P⇝O
- 例子：社交网站
  1. 用户A发表评论
  2. 用户B发表评论对A的评论进行反驳
  - 这两个评论操作之间**存在因果关系** → 时间先后关系
- **因果一致性**：解决操作乱序
  - 若一个客户端**观察到某个写操作**的执行结果，那么该客户端必须**也要能观察到发生在这个操作之前的且有因果关系的**其他写操作的执行结果

分布式系统中的事件顺序

- 一般不存在统一的物理时钟
  - 时钟漂移
  - 网络延时不确定
- 事件先后
  - 同一进程执行先后
  - 不同进程根据消息确定先后
- Lamport时钟
  - 逻辑时钟



因果一致性是四种属性的交集

- 单调读（Monotonic reads）
  - 同一客户端中，后续读到的值一定不能比前面读的值陈旧
- 单调写（Monotonic writes）
  - 其他客户端所读到的一个客户端中写操作的执行顺序与该客户端写操作的真实执行顺序一致
- 写后读（Read your writes）
  - 同一客户端中，写之后读到的数据不能比写的数据陈旧
- 读后写（Writes Follow Reads）
  - 同一客户端中，读到某个写操作a的结果之后再写b，那么其余客户端若能读到b的结果，其必须至少要能读到a的结果

- 不满足，则可能违反用户因果常识

因果一致的收敛性

- 收敛性
  - 一个数据项的各副本最终会收敛到同一状态
- 因果一致的收敛性
  - 最终会收敛到不违反因果一致性的同一状态
- 例子
  - 若 w(x=1) ⇝ w(x=2)，则x的状态不会收敛至1

保障因果一致性的要素

- 写操作的标识
  - <数据项的键, 逻辑时间戳>二元组
- 因果关系的存储及传递
  - 每个写操作所依赖写操作的标识也会被持久化
  - 每存取一个数据，客户端便保存对应写操作的标识
- 因果关系检查
  - 客户端读取数据时，需要根据本地记录确定不违反因果关系的逻辑时间戳 ts
  - 只有所获取数据的时间戳不比 ts 小才能通过检查
  - 否则，执行重读操作，直至读取到能通过检查的数据

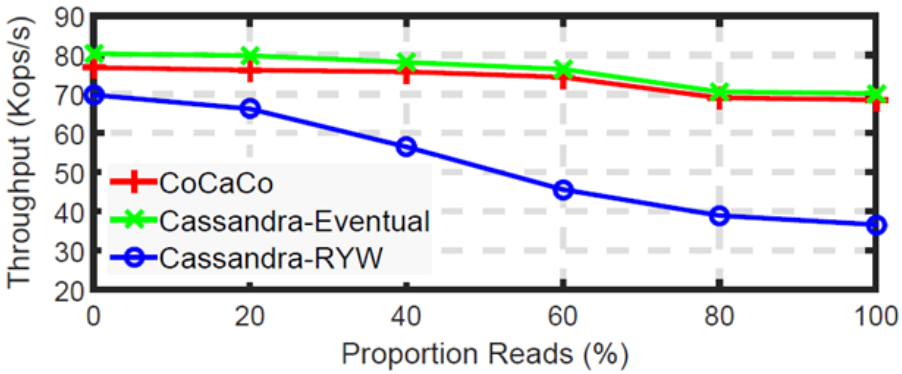
因果一致性保证

- 违反因果一致性的读操作比例

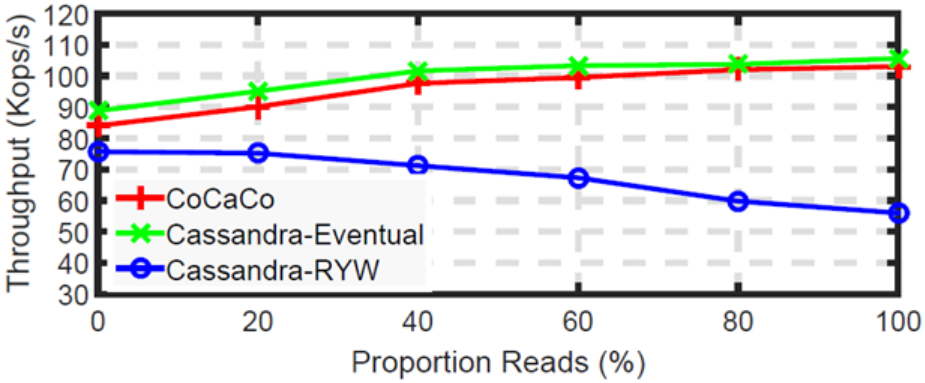
	读操作在所有操作中的比例			
	20%	40%	60%	80%
Cassandra-Eventual	0.16%	0.08%	0.03%	0.02%
Cassandra-RYW	>0	>0	>0	>0
CoCaCo	0	0	0	0

Cassandra: 分布式数据服务。
因果一致性系统CoCaCo未出现违反的情况

因果一致性对性能影响



(a) Uniform Distribution



(b) Zipfian Distribution

Q&A