



第九届中国数据库技术大会
DATABASE TECHNOLOGY CONFERENCE CHINA 2018

地域分布式系统设计与实践

李明昊

DTCC
2018

2018.05.10 – 12 北京国际会议中心



IT168.com

ChinaUnix

ITPUB

主题：

在一个复杂的软硬件环境中，设计一个分布式存储系统遇到了哪些问题？是如何解决的？

一些经验的总结，供参考

1. 提出问题

2. 分析问题

3. 解决问题

4. 经验总结

1. 提出问题

2. 分析问题

3. 解决问题

4. 经验总结

1. 提出问题

万达在全国具有200+的实体广场，每个广场具备自己的内部机房。包括路由器，无线热点，主机，防火墙等硬件资源。当用户进入广场内部时，手机会通过热点自动连入广场内部局域网系统。

这些广场机房的服务质量远达不到专业数据中心的服务质量。广场内部接近局域网环境，广场之间接近广域网环境。

如果我们能有效的利用广场内部硬件、网络资源，将用户或商品的一部分数据存入广场内部机房，并做合适的计算加工，用户将享受到本地计算与本地存储的体验，同时降低了硬件成本。但必须想办法解决小机房服务质量不达标的问题。

1. 提出问题

2. 分析问题

3. 解决问题

4. 经验总结

2. 分析问题

2.1 了解必要的理论体系

- 历史脉络
- ACID、CAP 两个C有什么不同
- 如何划清问题边界？
- 关于CAP的误解

2.2 实地场测，抽象问题模型

2.3 协议选型与改造



2. 分析问题

2.1 了解必要的理论体系

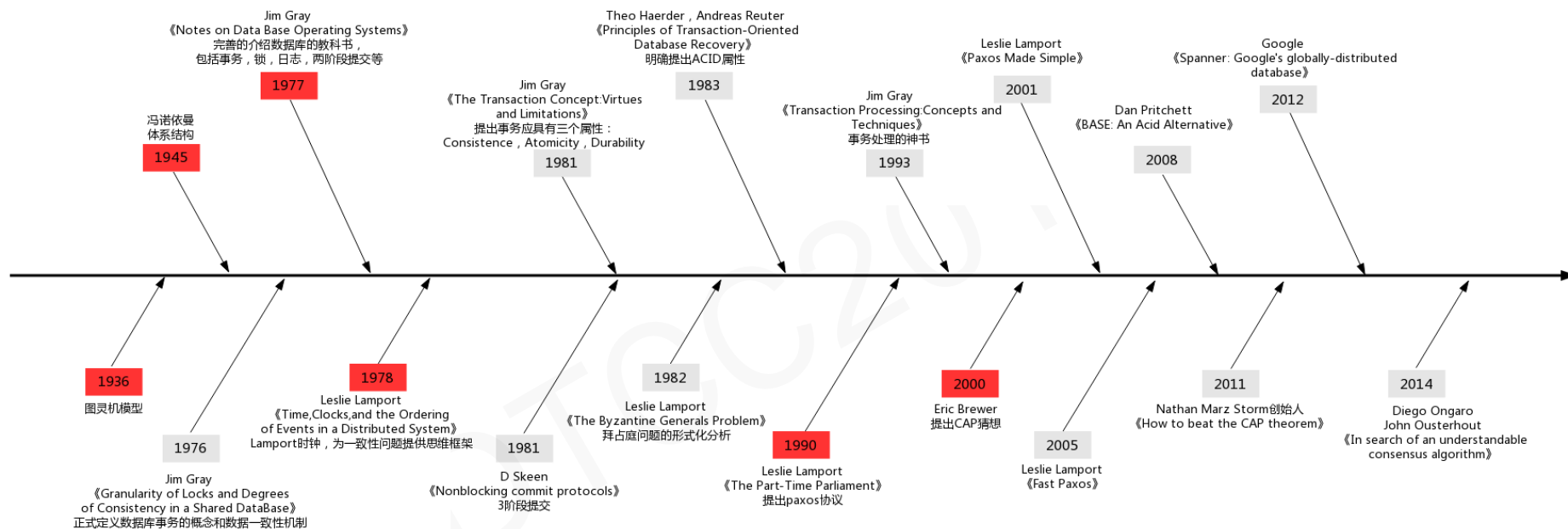
- 历史脉络
- ACID、CAP 两个C有什么不同
- 如何划清问题边界？
- 关于CAP的误解

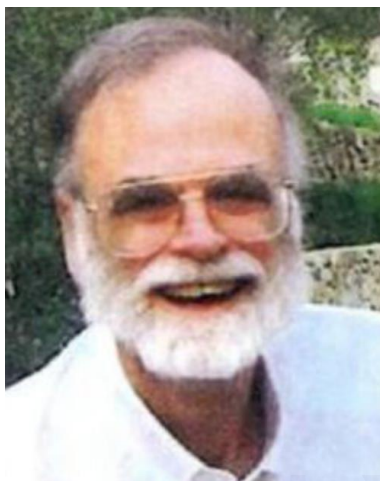
2.2 实地场测，抽象问题模型

2.3 协议选型与改造



历史脉络



[+](#) | [★ 收藏](#) | [👍 13](#) | [🔗](#)

詹姆斯·尼古拉·格雷

詹姆斯·尼古拉·格雷（James Nicholas "Jim" Gray, 1944年—），美国资讯工程学家。他就读于柏克莱加州大学，1966年取得工程数学学士，1969年取得计算机科学博士。他曾于IBM、天登电脑公司和迪吉多工作。1995年成为微软研究员。他有份开发的数据库和交易处理系统有IBM的System R、微软的Terraserver和Skyserver。他提出了资料方块、锁定颗粒等概念。他亦有份开发Windows Live Local。2007年，他独自航向法拉伦岛，打算撒散母亲的骨灰，1月28日，他的船失踪了。2月1日，DigitalGlobe扫描过一带，产生了上千张影象。影象放于Amazon Mechanical Turk，希望人们能合力找出他的船。2月16日，他的家人要求取消搜索行动。 [1]

中文名	詹姆斯·尼古拉·格雷	职业	工程师
外文名	James Nicholas "Jim" Gray	毕业院校	美国加州大学
国 籍	美国	主要成就	图灵奖

人物经历

1960年毕业于麻省理工学院数学专业。

1963年获得布兰迪斯大学数学硕士学位。

1965-1969年任教于马尔波罗学院

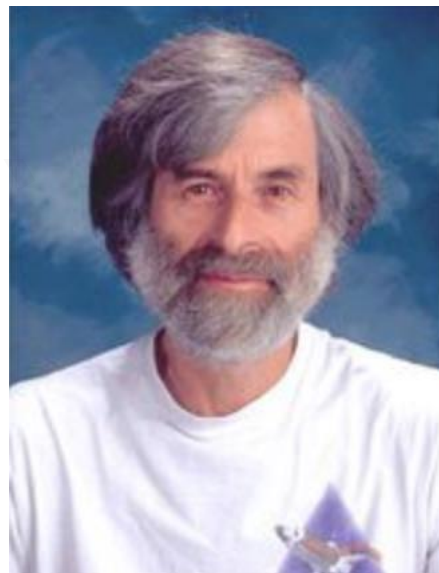
1970-1972年，麻省计算机协会系统设计员。

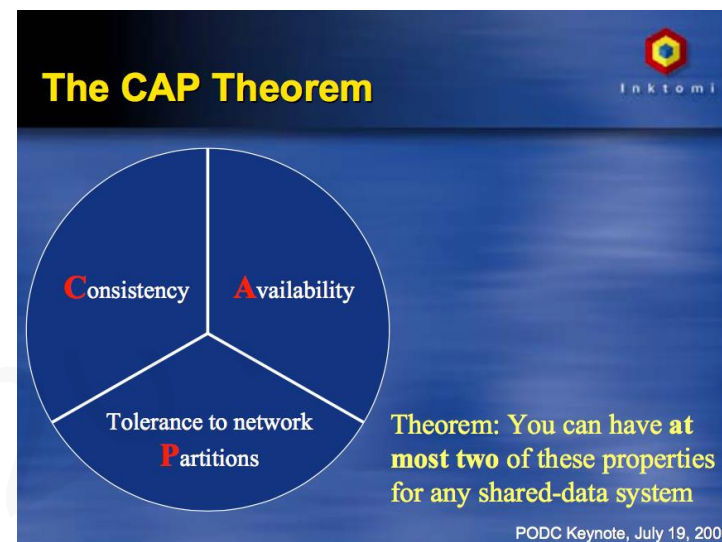
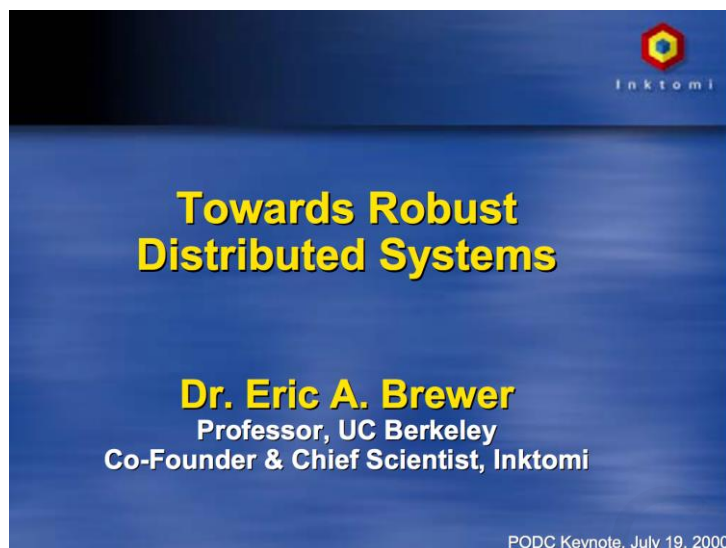
1972年获得布兰迪斯大学数学博士学位。

1972-1977年，麻省计算机协会研究员。

1977-1985年，SRI 公司计算机科学实验室研究员。

Leslie Lamport






2000 Eric Brewer UC Berkeley 提出CAP猜想

2002 Seth Gilbert, Nancy Lynch MIT 证明CAP

2011 Nathan Marz Twitter首席工程师, Storm创始人

《How to beat the CAP theorem》

2012 Eric Brewer 《How the "Rules" Have Changed》


Inktomi

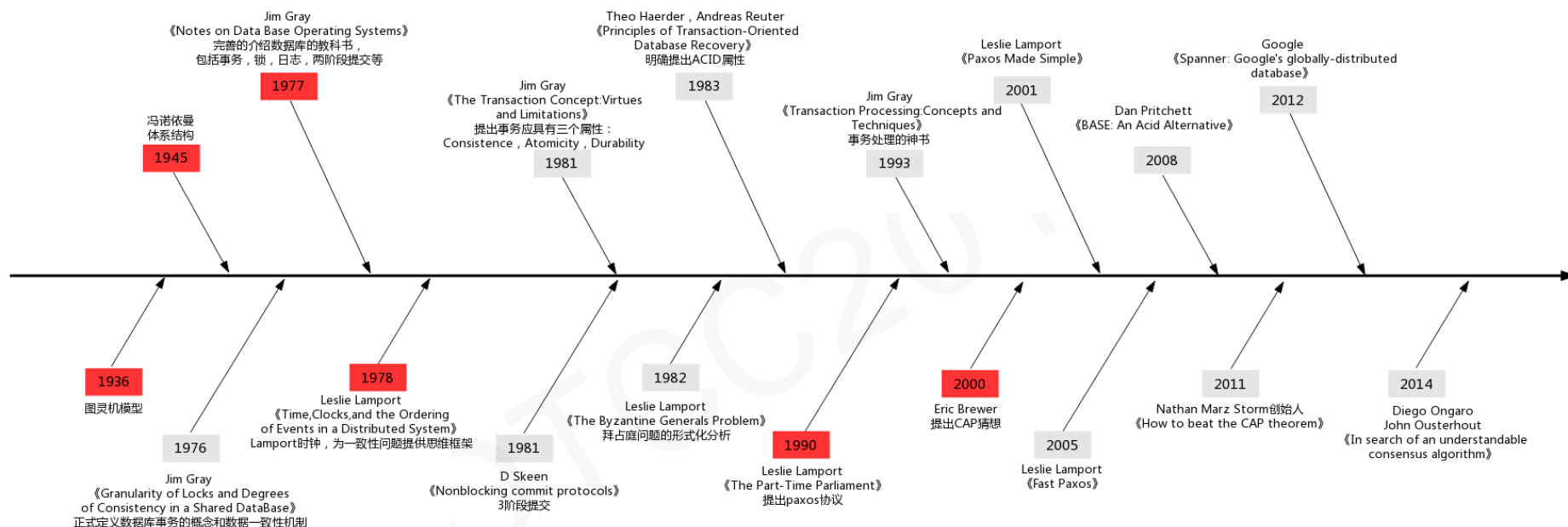
ACID vs. BASE

<u>ACID</u>	<u>BASE</u>
◆ Strong consistency	◆ Weak consistency <ul style="list-style-type: none">— stale data OK
◆ Isolation	◆ Availability first
◆ Focus on “commit”	◆ Best effort
◆ Nested transactions	◆ Approximate answers OK
◆ Availability?	◆ Aggressive (optimistic)
◆ Conservative (pessimistic)	◆ Simpler!
◆ Difficult evolution (e.g. schema)	◆ Faster
	◆ Easier evolution

← But I think it's a spectrum →

PODC Keynote, July 19, 2000

历史脉络



1. 计算机与互联网技术起源于战争的需要
2. 以Jim Gray为代表的数据库技术和 以Leslie Lamport为代表的分布式技术逐渐融合

2. 分析问题

2.1 了解必要的理论体系

- 历史脉络
- **ACID、CAP 两个C有什么不同**
- 如何划清问题边界？
- 关于CAP的误解

2.2 实地场测，抽象问题模型

2.3 协议选型与改造



ACID

CAP

DTCC2018

ACID

CAP

Consistency Consensus

DTCC2018



ACID

CAP

Consistency Consensus

始终如一的 共识的

Consistency

Principles of Transaction-Oriented Database Recovery

THEO HAERDER

Fachbereich Informatik, University of Kaiserslautern, West Germany

ANDREAS REUTER¹

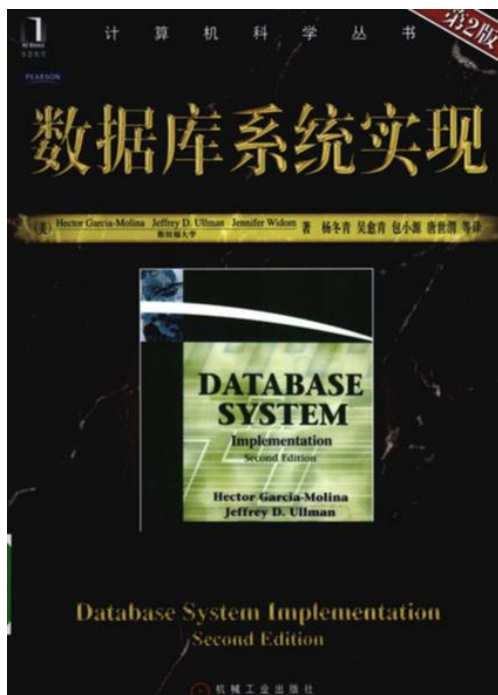
IBM Research Laboratory, San Jose, California 95193



Consistency

Consistency. A transaction reaching its normal end (EOT, end of transaction), thereby committing its results, preserves the consistency of the database. In other words, each successful transaction by definition commits only legal results. This con-

Consistency



事务的 ACID 特性

通常我们说正确执行的事务符合“ACID 准则”，其中：

- “A”表示“原子性”(Atomicity)，即事务完全执行或完全不执行。
- “I”表示“孤立性”(Isolation)，即表面看起来每一个事务都是在没有其他事务同时执行的情况下执行的。
- “D”表示“持久性”(Durability)，即一旦事务完成了，则事务对数据库的影响就不会丢失。

剩下的一个字母“C”表示“一致性”(Consistency)。也就是说，所有的数据库都有一致性约束，或关于数据之间联系的预期状况(例如，在一个事务完成后，账户余额不能是负数)。期望事务能够保持数据库的一致性。



Consensus

In Search of an Understandable Consensus Algorithm (Extended Version)

Diego Ongaro and John Ousterhout
Stanford University

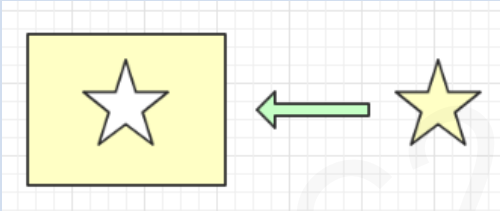

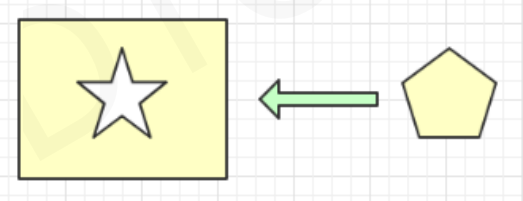

Consensus

1 Introduction

Consensus algorithms allow a collection of machines to work as a coherent group that can survive the failures of some of its members. Because of this, they play a key role in building reliable large-scale software systems. Paxos [15, 16] has dominated the discussion of consensus algorithms over the last decade: most implementations of **consensus** are based on Paxos or influenced by it, and Paxos has become the primary vehicle used to teach students about **consensus**.



两个C的不同

	Consistency	Consensus
YES !!		
NO !!		

关于分布式领域里的“一致性”，又可以进一步细分

All Things Distributed

Werner Vogels' weblog on building scalable and robust distributed systems.

Eventually Consistent

By Werner Vogels on 19 December 2007 02:03 PM | [Permalink](#) | [Comments \(\)](#)

I wrote a first version of this posting on consistency models in December 2007, but I was never happy with it as it was written in haste and the topic is important enough to receive a more thorough treatment. ACM Queue asked me to revise it for use in their magazine and I took the opportunity to improve the article.

I posted an update to this article in December 2008 under the title [Eventually Consistent - Revisted](#). - please read that article instead of this one. I am leaving this one here for transparency/historical reasons and because the comments helped me improve the article. For which I am grateful



Contact Info

Werner Vogels
CTO - Amazon.com

werner@allthingsdistributed.com

Recently there has been a lot of discussion about the concept of *eventual consistency* in the context of data replication. In this positing I would like to try to collect some of the principles and abstractions related to large scale data replication and the trade-offs between high-availability and data consistency. I consider this work-in-progrees as I don't expect to get every definition criso the first time.

两个C的不同

- **Strong consistency.** After the update completes, any subsequent access (by A, B, or C) will return the updated value.
- **Weak consistency.** The system does not guarantee that subsequent accesses will return the updated value. A number of conditions need to be met before the value will be returned. The period between the update and the moment when it is guaranteed that any observer will always see the updated value is dubbed the *inconsistency window*.
- **Eventual consistency.** This is a specific form of weak consistency; the storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value. If no failures occur, the maximum size of the inconsistency window can be determined based on factors such as communication delays, the load on the system, and the number of replicas involved in the replication scheme. The most popular system that implements eventual consistency is DNS (Domain Name System). Updates to a name are distributed according to a configured pattern and in combination with time-controlled caches; eventually, all clients will see the update.



The eventual consistency model has a number of variations that are important to consider:

- **Causal consistency.** If process A has communicated to process B that it has updated a data item, a subsequent access by process B will return the updated value, and a write is guaranteed to supersede the earlier write. Access by process C that has no causal relationship to process A is subject to the normal eventual consistency rules.
- **Read-your-writes consistency.** This is an important model where process A, after it has updated a data item, always accesses the updated value and will never see an older value. This is a special case of the causal consistency model.
- **Session consistency.** This is a practical version of the previous model, where a process accesses the storage system in the context of a session. As long as the session exists, the system guarantees read-your-writes consistency. If the session terminates because of a certain failure scenario, a new session needs to be created and the guarantees do not overlap the sessions.
- **Monotonic read consistency.** If a process has seen a particular value for the object, any subsequent accesses will never return any previous values.
- **Monotonic write consistency.** In this case the system guarantees to serialize the writes by the same process. Systems that do not guarantee this level of consistency are notoriously hard to program.

2. 分析问题

2.1 了解必要的理论体系

- 历史脉络
- ACID、CAP 两个C有什么不同
- 如何划清问题边界?
- 关于CAP的误解

2.2 实地场测，抽象问题模型

2.3 协议选型与改造

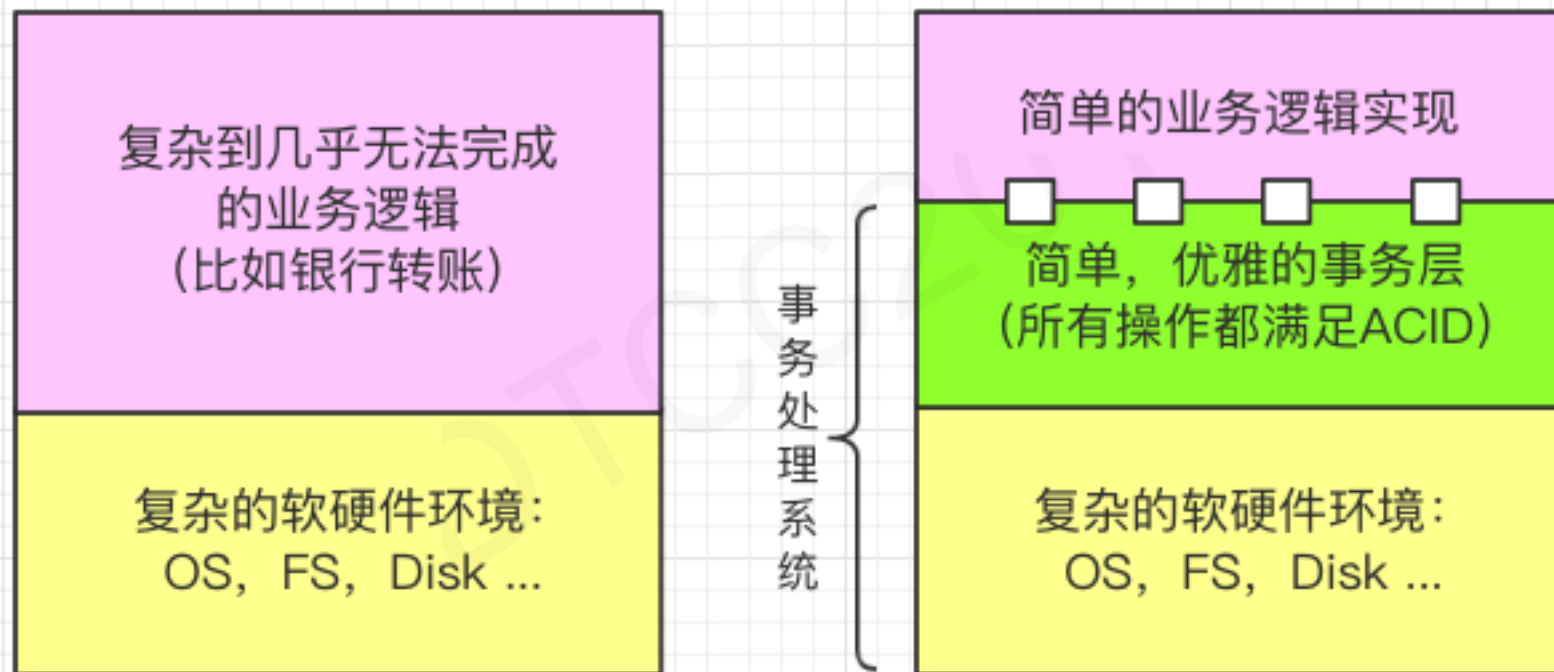
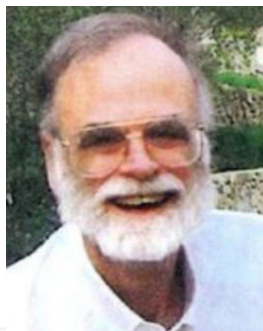


如何划清问题边界

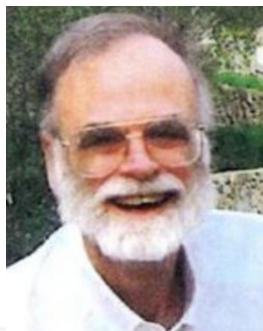
复杂到几乎无法完成的
业务逻辑
(比如银行转账)

复杂的软硬件环境：
OS, FS, Disk ...

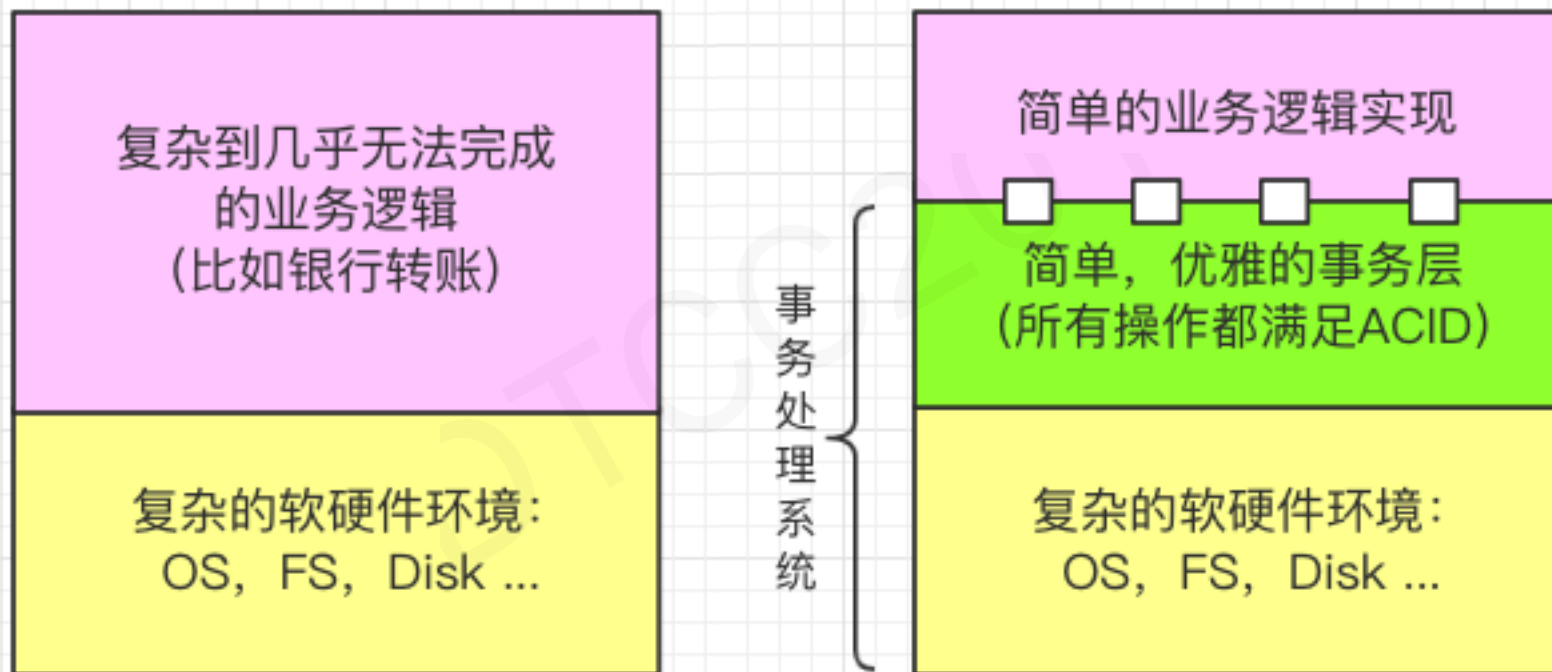
如何划清问题边界



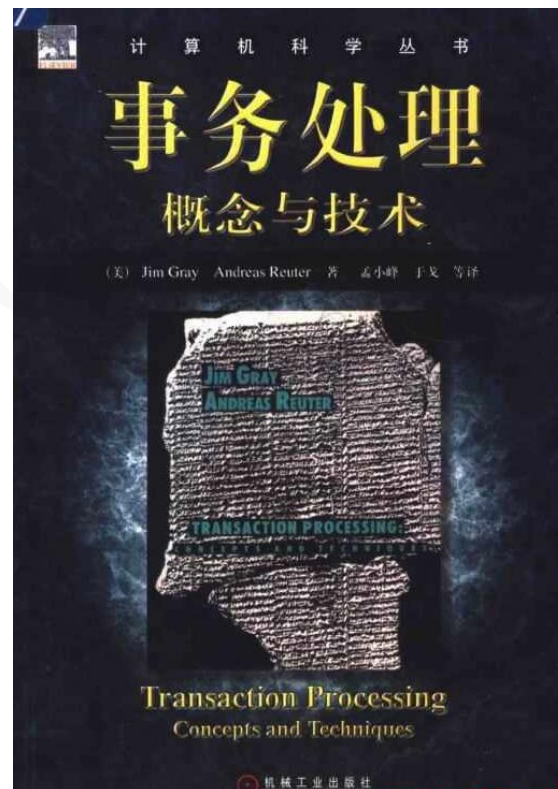
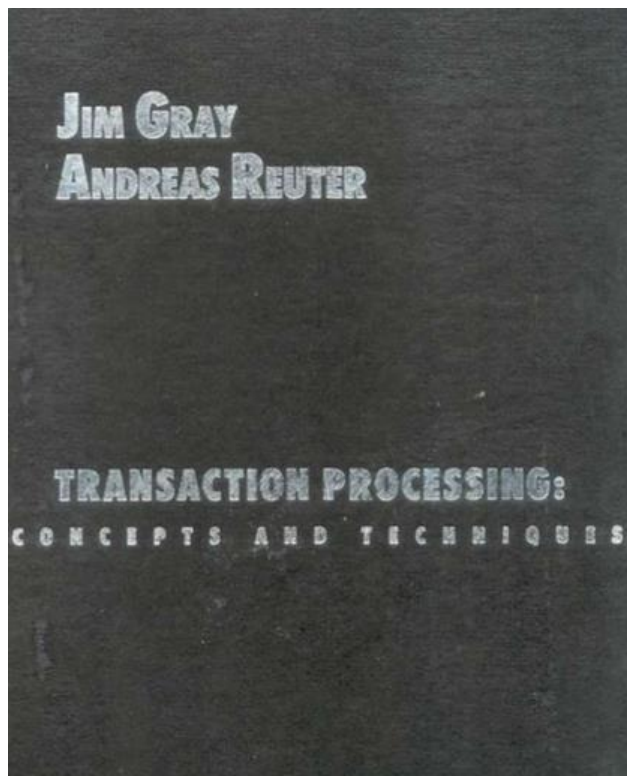
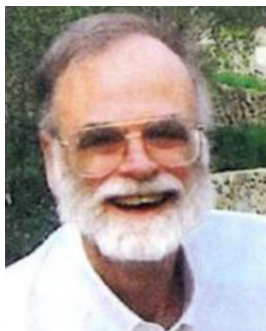
如何划清问题边界



业务逻辑简单了，
但ACID又如何保证呢？



如何划清问题边界



DTCC
2018

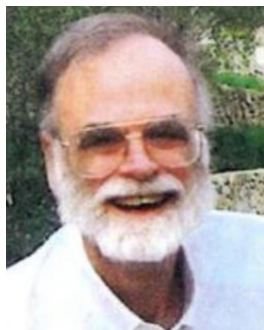
数领先机 智赢未来 (9)

IT168.com

ChinaUnix

ITPUB





3.1.1 简单概率概述

一个事件A在一定时间内发生的概率用 $P(A)$ 表示。概率值在0和1之间。概率值为0表示事件从不发生；1表示事件肯定发生。事件A不发生的概率是 $1 - P(A)$ 。如果A和B之间一个事件发生不会影响另一个事件发生的概率，那么A和B就是独立事件 (independent event)。给定两个独立事件A和B，考虑下面的可能情况和它们的计算公式：

两个事件都发生 A和B都发生的概率是它们各自概率的乘积：

$$P(A \text{ and } B) = P(A) \cdot P(B) \quad (3-1)$$

至少有一个事件发生 A和B至少一个发生的概率是：A发生的概率加上A不发生的概率与B发生的概率的乘积：

$$\begin{aligned} P(A \text{ or } B) &= P(A) + (1 - P(A)) \cdot P(B) \\ &= P(A) + P(B) - P(A) \cdot P(B) \end{aligned} \quad (3-2)$$

式(3-2)中有一小项 $P(A) \cdot P(B)$ ，如果 $P(A)$ 和 $P(B)$ 都很小，这一项就可以忽略

$$\approx P(A) + P(B) \text{ 如果 } P(A) \ll 1, \text{ 并且 } P(B) \ll 1 \quad (3-3)$$

例如，如果一个系统一天中发生故障的可能性是0.01，电话网络发生故障的可能性是0.02，终端发生故障的可能性是0.03，那么，三个都发生故障的可能性是 $0.01 \cdot 0.02 \cdot 0.03$ ，也就是 6×10^{-6} 。三个中任一个发生故障的可能性是 $0.01 + 0.02 + 0.03$ ，也就是0.06。

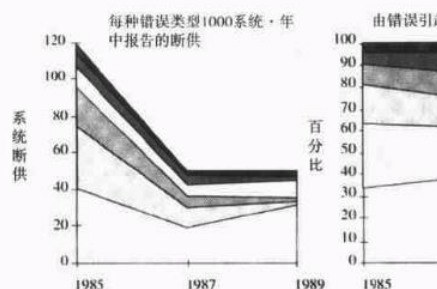


图3-7 表明断供频率的衰退和每一类系统断供所占份额因从硬件和维护转向了软件和操作，转向操作/到环境和操作断供的数据还有所保留

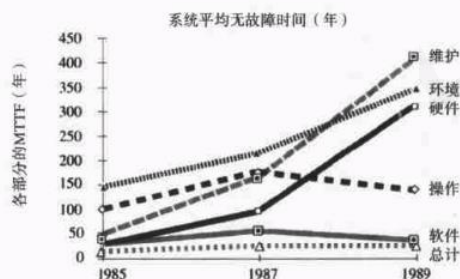
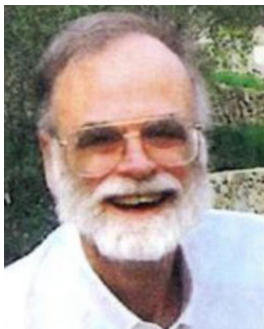


图3-8 表明了由致命原因引起的系统平均故障发生时间的变化趋势。注意，软件和操作保持不变，而硬件、维护和环境得到戏剧性的改善



3.1.1 简单概率概述

一个事件A在一定时间内发生的概率用 $P(A)$ 表示。概率值在0和1之间。概率值为0表示事件从不发生；1表示事件肯定发生。事件A不发生的概率是 $1 - P(A)$ 。如果A和B之间一个事件发生不会影响另一个事件发生的概率，那么A和B就是独立事件 (independent event)。给定两个独立事件A和B，考虑下面的可能情况和它们的计算公式：

两个事件都发生 A和B都发生的概率是它们各自概率的乘积：

$$P(A \text{ and } B) = P(A) \cdot P(B) \quad (3-1)$$

至少有一个事件发生 A和B至少一个发生的概率是：A发生的概率加上A不发生的概率与B发生的概率的乘积：

$$\begin{aligned} P(A \text{ or } B) &= P(A) + (1 - P(A)) \cdot P(B) \\ &= P(A) + P(B) - P(A) \cdot P(B) \end{aligned} \quad (3-2)$$

式(3-2)中有一小项 $P(A) \cdot P(B)$ ，如果 $P(A)$ 和 $P(B)$ 都很小，这一项就可以忽略

$$\approx P(A) + P(B) \text{ 如果 } P(A) \ll 1, \text{ 并且 } P(B) \ll 1 \quad (3-3)$$

例如，如果一个系统一天中发生故障的可能性是0.01，电话网络发生故障的可能性是0.02，终端发生故障的可能性是0.03，那么，三个都发生故障的可能性是 $0.01 \cdot 0.02 \cdot 0.03$ ，也就是 6×10^{-6} 。三个中任一个发生故障的可能性是 $0.01 + 0.02 + 0.03$ ，也就是0.06。

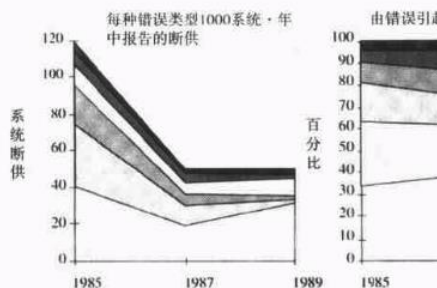


图3-7 表明断供频率的衰退和每一类系统断供所占份额因从硬件和维护转向了软件和操作，转向操作和环境断供的数据也有所保留

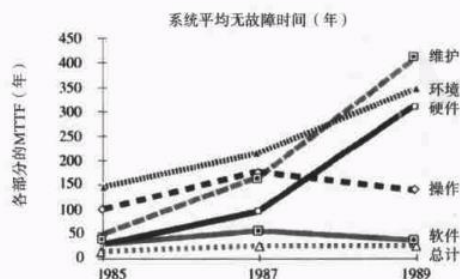


图3-8 表明了由致命原因引起的系统平均故障发生时间的变化趋势。注意，软件和操作保持不变，而硬件、维护和环境得到戏剧性的改善

绝对的ACID是不存在的， 只是在某种概率上得到了保证

基于假设解决问题

问题域（软件、硬件、网络...）

----- 合理假设

需要重点关注

可以适当忽略

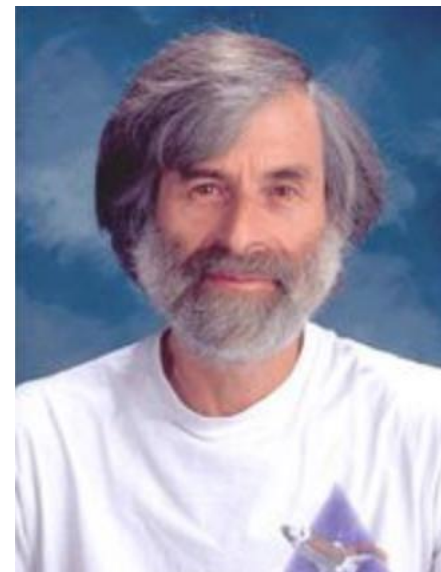


分布式领域里的模型假设

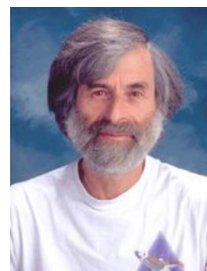
Paxos Made Simple

Leslie Lamport

01 Nov 2001



分布式领域里的模型假设



Assume that agents can communicate with one another by sending messages. We use the customary asynchronous, non-Byzantine model, in which:

- Agents operate at arbitrary speed, may fail by stopping, and may restart. Since all agents may fail after a value is chosen and then restart, a solution is impossible unless some information can be remembered by an agent that has failed and restarted.
- Messages can take arbitrarily long to be delivered, can be duplicated, and can be lost, but they are not corrupted.



分布式领域里的模型假设

1. 网络中包含多个主机
2. 每个主机包含若干进程与磁盘
3. 进程间通过网络包通信
4. 网络包可能会重复，丢失，乱序，但内容不会出错
5. 进程可能会重启
6. 磁盘可能会损坏，导致数据丢失，但数据不会出错
7. RTT一般在1ms之内



分布式领域里的模型假设

局域网

1. 网络中包含多个主机
2. 每个主机包含若干进程与磁盘
3. 进程间通过网络包通信
4. 网络包可能会重复，丢失，乱序，但内容不会出错
5. 进程可能会重启
6. 磁盘可能会损坏，导致数据丢失，但数据不会出错
7. RTT一般在1ms之内

什么是拜占庭问题？

可以简单的理解为：承诺与实际表现不相符

DTCC2018

出现拜占庭问题怎么办？

1. 网络包数据出错
2. 磁盘数据出错
3. 程序出现bug

。 。 。

DTCC2018



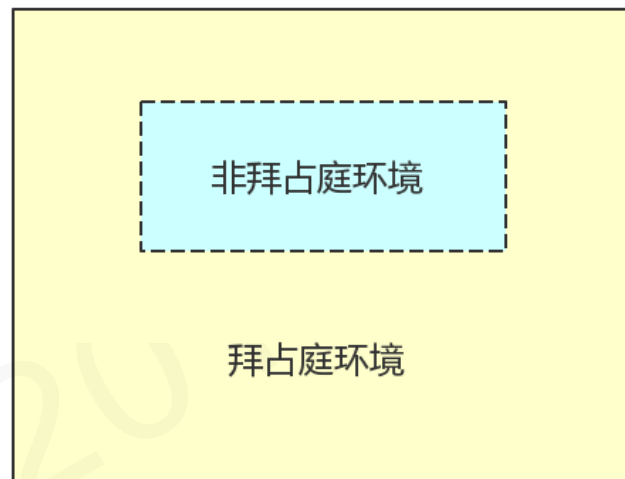
出现拜占庭问题怎么办？

1. 网络包数据出错
 2. 磁盘数据出错
 3. 程序出现bug
- 。 。 。

通过其他方式在一定的概率上解决拜占庭问题，
但不要影响我们基于非拜占庭环境的架构设计

出现拜占庭问题怎么办？

1. 网络包数据出错
 2. 磁盘数据出错
 3. 程序出现bug
- 。 。 。



通过其他方式在一定的概率上解决拜占庭问题，
但不要影响我们基于非拜占庭环境的架构设计

2. 分析问题

2.1 了解必要的理论体系


- 历史脉络
- ACID、CAP 两个C有什么不同
- 如何划清问题边界？
- 关于CAP的误解

2.2 实地场测，抽象问题模型

2.3 协议选型与改造




关于CAP的误解



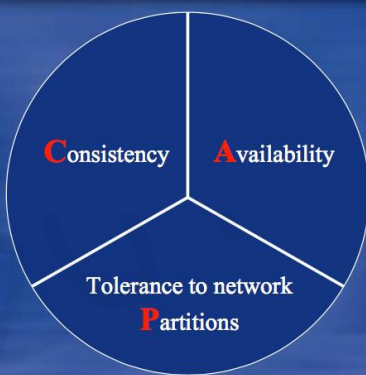
Towards Robust Distributed Systems

Dr. Eric A. Brewer
Professor, UC Berkeley
Co-Founder & Chief Scientist, Inktomi

PODC Keynote, July 19, 2000



The CAP Theorem



Theorem: You can have at most two of these properties for any shared-data system

PODC Keynote, July 19, 2000

2000 Eric Brewer UC Berkeley 提出CAP猜想

2002 Seth Gilbert, Nancy Lynch MIT 证明CAP

2011 Nathan Marz Twitter首席工程师, Storm创始人

《How to beat the CAP theorem》

2012 Eric Brewer 《How the "Rules" Have Changed》

CA or AP or CP ?

DTCC2018



CA or AP or CP ?

NO!!



$$0.9 C + 0.9 A + 0.2 P$$

$$0.7 C + 0.7 A + 0.6 P$$

DTCC2018



$$0.9 C + 0.9 A + 0.2 P$$

$$0.7 C + 0.7 A + 0.6 P$$

可用性：在可接受的时间内给出反馈结果。



$$0.9 C + 0.9 A + 0.2 P$$

$$0.7 C + 0.7 A + 0.6 P$$

不用太纠结，以满足实际需求为标准

2. 分析问题

2.1 了解必要的理论体系

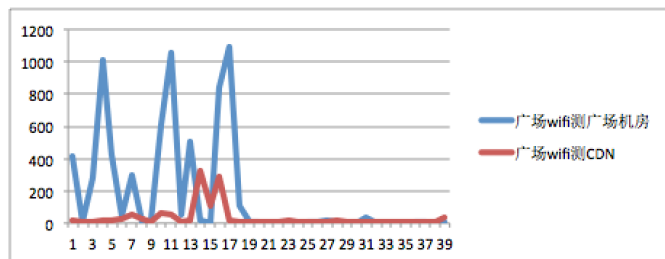
- 历史脉络
- ACID、CAP 两个C有什么不同
- 如何划清问题边界？
- 关于CAP的误解

2.2 实地场测，抽象问题模型

2.3 协议选型与改造



实地场测，抽象问题模型



	300-500B	5-7K	50-70K	370-390K	2.2M
首屏(ms)	276.8	271.2	269.3	264.9	268.8
页面加载完成(ms)	494.0	498.8	536.1	905.3	2481.9
接口加载时间(ms)	130.4	121.5	127.9	124.1	125.5
图片 1 加载时间(ms)	19.8	19.0	54.8	377.4	661.3
图片 2 加载时间(ms)	32.2	37.3	72.7	408.2	1792.1
图片 3 加载时间(ms)	43.4	51.7	88.1	431.2	1859.1
图片 4 加载时间(ms)	57.3	65.9	100.1	466.3	1981.0

	平均耗时		中位数		平均方差		稳定性
	去噪前	去噪后	去噪前	去噪后	去噪前	去噪后	
Wifi 广场	3.588ms	2.83ms	2.31ms	2.25ms	3.05	1.62	较稳定
Wifi 成都	307.05ms	254.64ms	187ms	183ms	207.82	119.66	不稳定
4g 广场	139.66ms	125.9ms	128.5ms	124ms	33.95	21	稳定
4g 成都	119.67ms	90.5ms	94.75ms	90.55ms	64.57	9.13	稳定

分布式领域里的模型假设

1. 网络中包含多个主机
2. 每个主机包含若干进程与磁盘
3. 进程间通过网络包通信
4. 网络包可能会重复，丢失，乱序，但内容不会出错
5. 进程可能会重启
6. 磁盘可能会损坏，导致数据丢失，但数据不会出错
7. RTT同广场1ms以下，同城10ms，跨城市几十ms

实地场测，抽象问题模型

1. 同城接近局域网
2. 跨城接近广域网
3. 供电系统弱于专业机房，机房断电概率高于专业机房
4. 冷却系统弱于专业机房，机器重启概率高于专业机房

2. 分析问题

2.1 了解必要的理论体系

- 历史脉络
- ACID、CAP 两个C有什么不同
- 如何划清问题边界？
- 关于CAP的误解

2.2 实地场测，抽象问题模型

2.3 协议选型与改造



Leslie Lamport

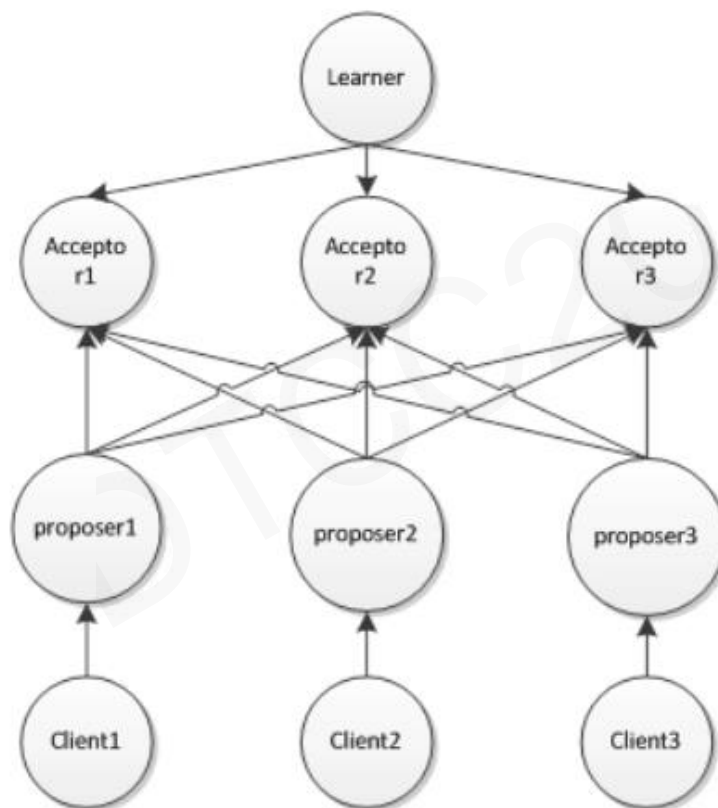
《Time,Clocks,and the Ordering of Events in a Distributed System》

《The Byzantine Generals Problem》

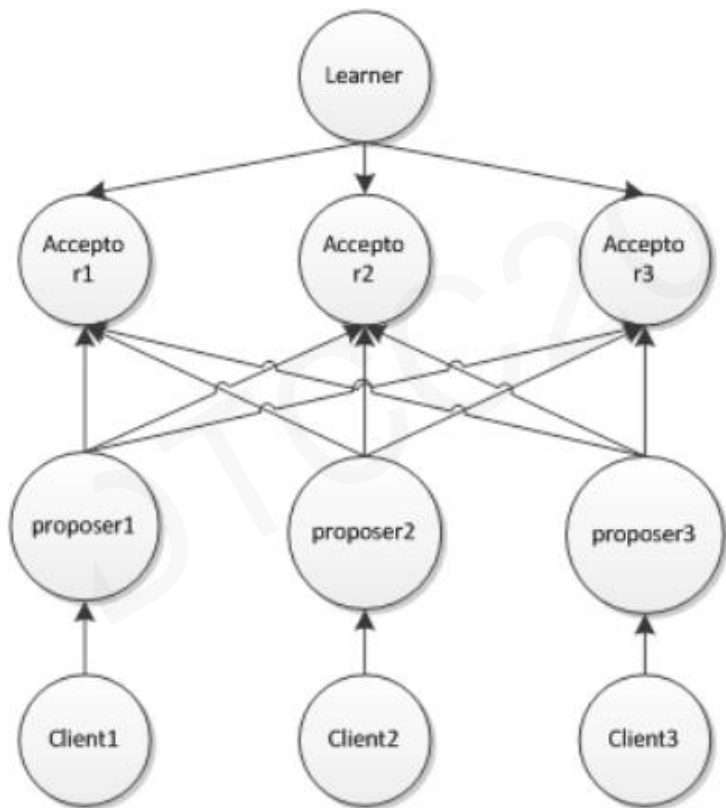
《The Part-Time Parliament》

《Paxos Made Simple》

多个节点如何就一个值达成一致？



多个节点如何就一个值达成一致？
value取得**大多数**acceptor的认可



多个节点就一个值达成一致有什么用？

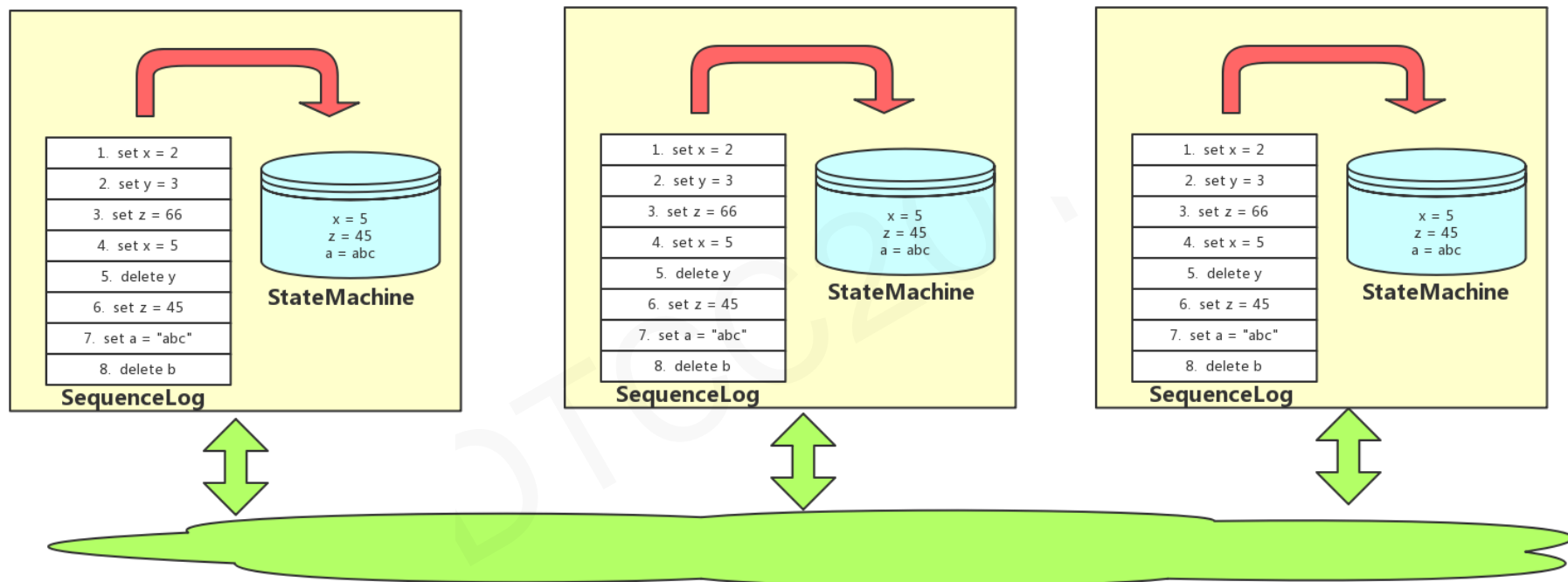
DTCC2018

多个节点就一个值达成一致有什么用？

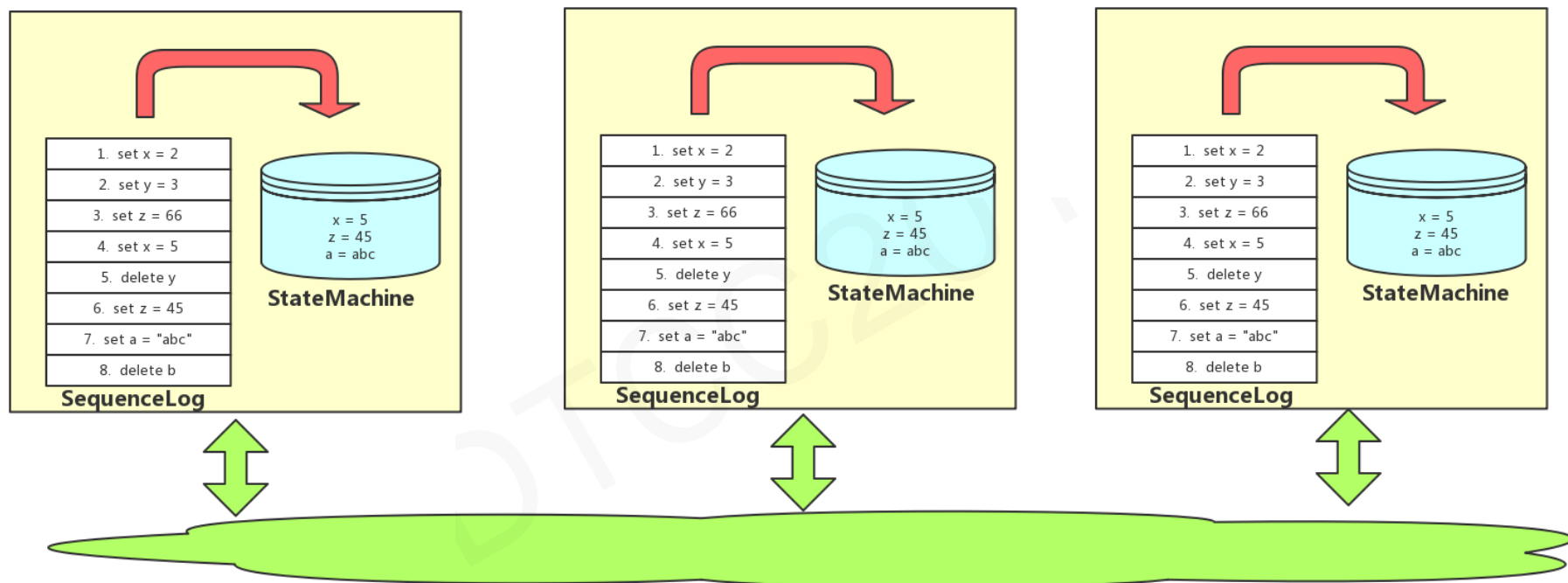
多个节点选主

DTCC2018

多个节点就一个值达成一致有什么用？



多个节点就一个值达成一致有什么用？



Multi-Paxos

Multi-Paxos 只是一个思路，很多细节论文里都没有描述，
导致系统实现起来困难重重

DTCC2018

Multi-Paxos 只是一个思路，很多细节论文里都没有描述，导致系统实现起来困难重重

BURROWS, M. The Chubby lock service for loosely-coupled distributed systems. In *Proc. OSDI'06, Symposium on Operating Systems Design and Implementation* (2006), USENIX, pp. 335–350.

CHANDRA, T. D., GRIESEMER, R., AND REDSTONE, J. Paxos made live: an engineering perspective. In *Proc. PODC'07, ACM Symposium on Principles of Distributed Computing* (2007), ACM, pp. 398–407.

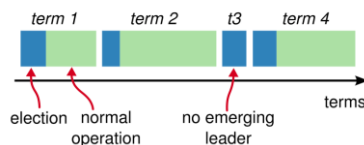
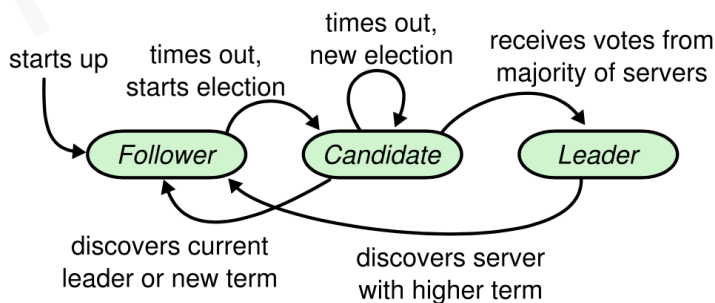
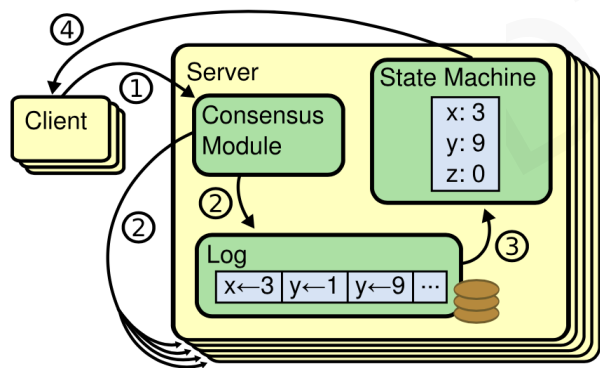
MAZIÈRES, D. Paxos made practical. <http://www.scs.stanford.edu/~dm/home/papers/paxos.pdf>, Jan. 2007.



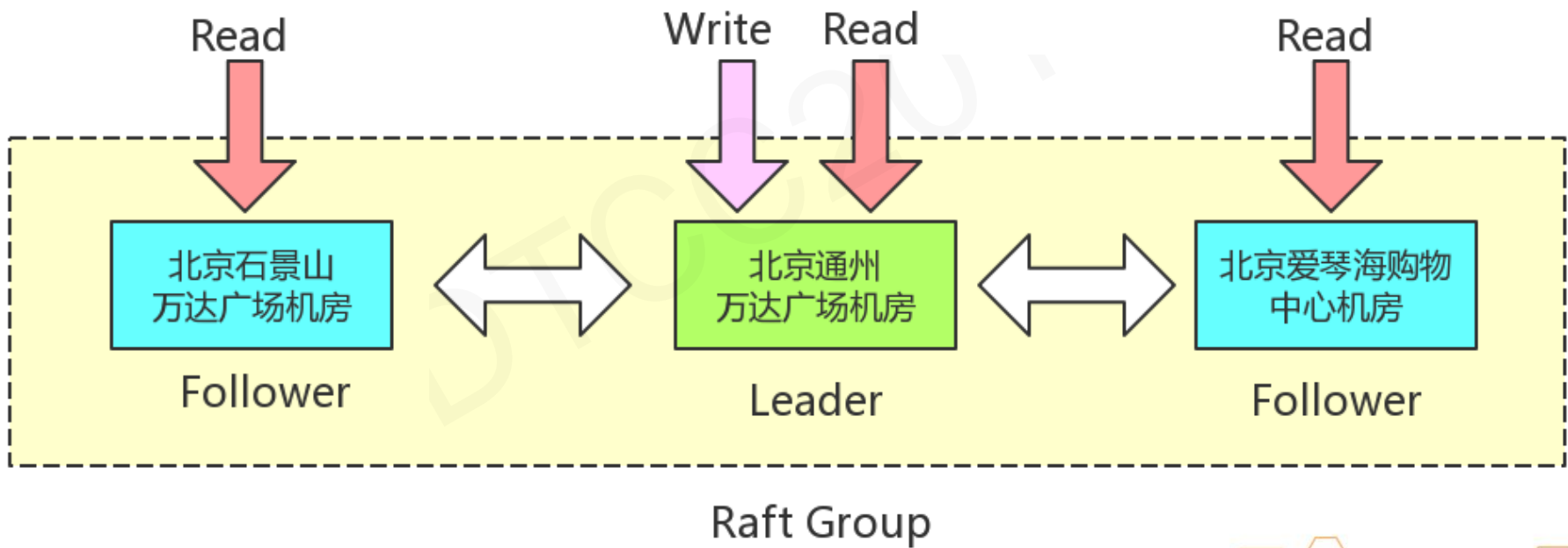
Multi-Paxos 只是一个思路，很多细节论文里都没有描述，导致系统实现起来问题不少

In Search of an Understandable Consensus Algorithm (Extended Version)

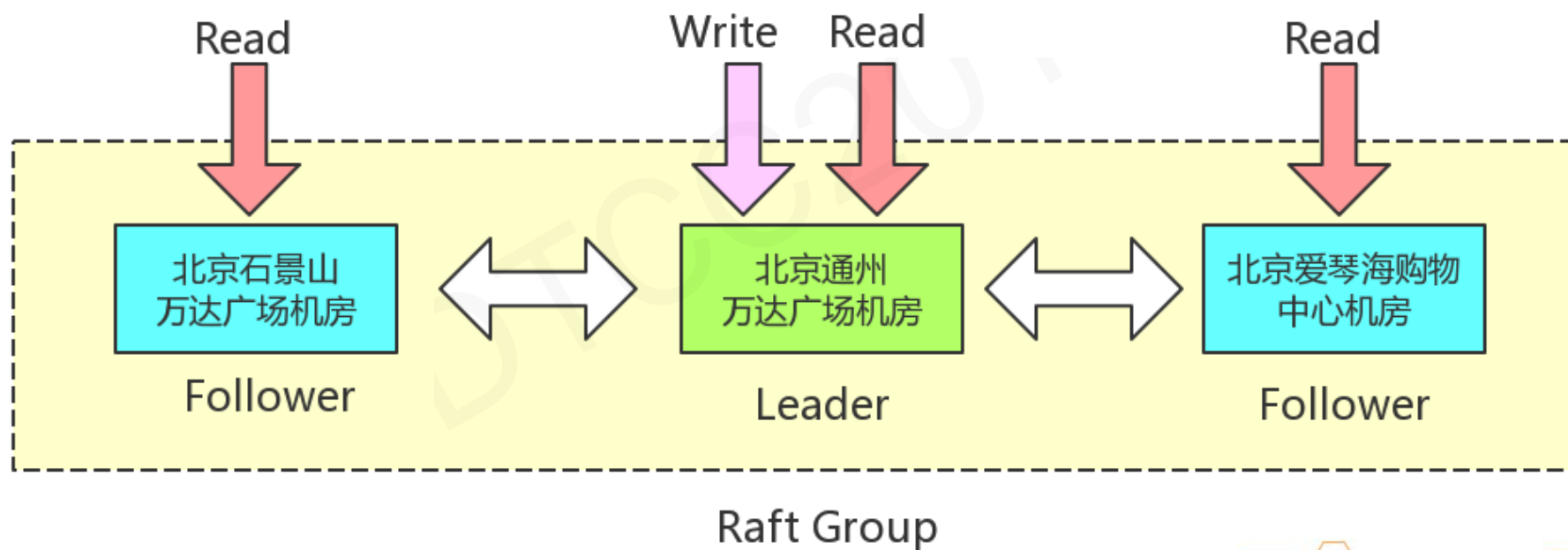
Diego Ongaro and John Ousterhout
Stanford University



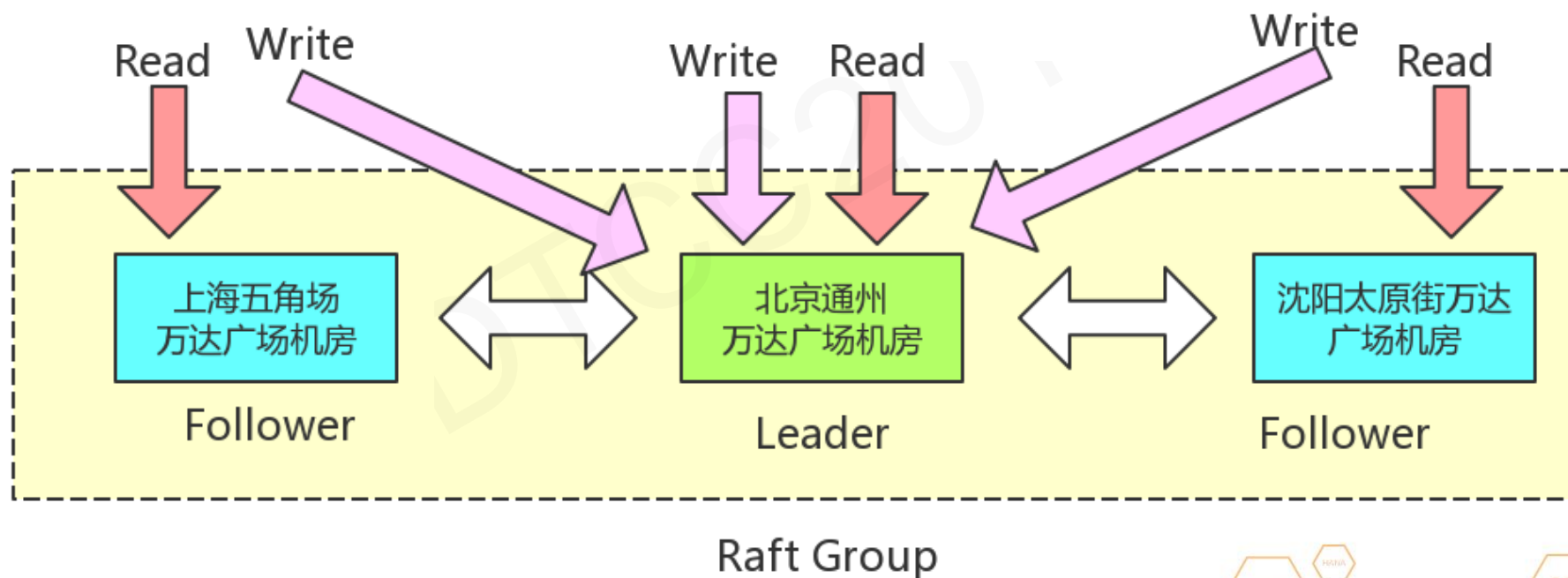
假设采用Paxos模型



假设采用Paxos模型，我们的系统会有什么问题？

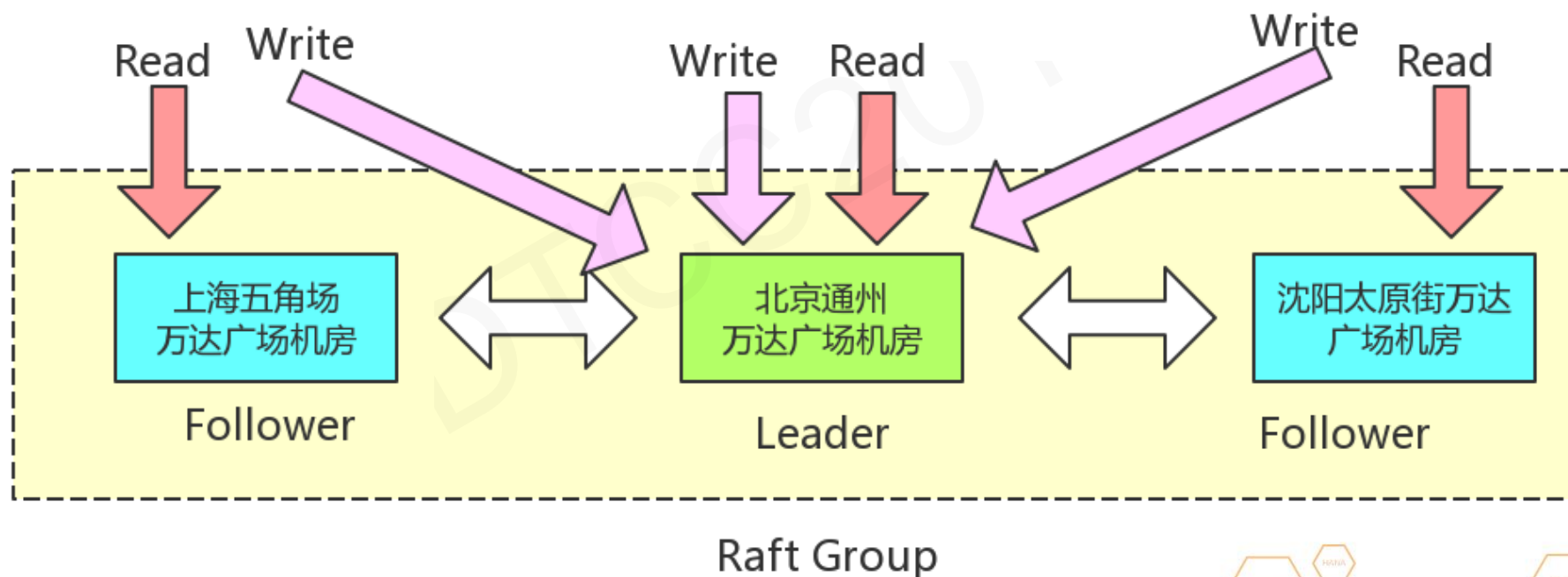


假设采用Paxos模型，我们的系统会有什么问题？

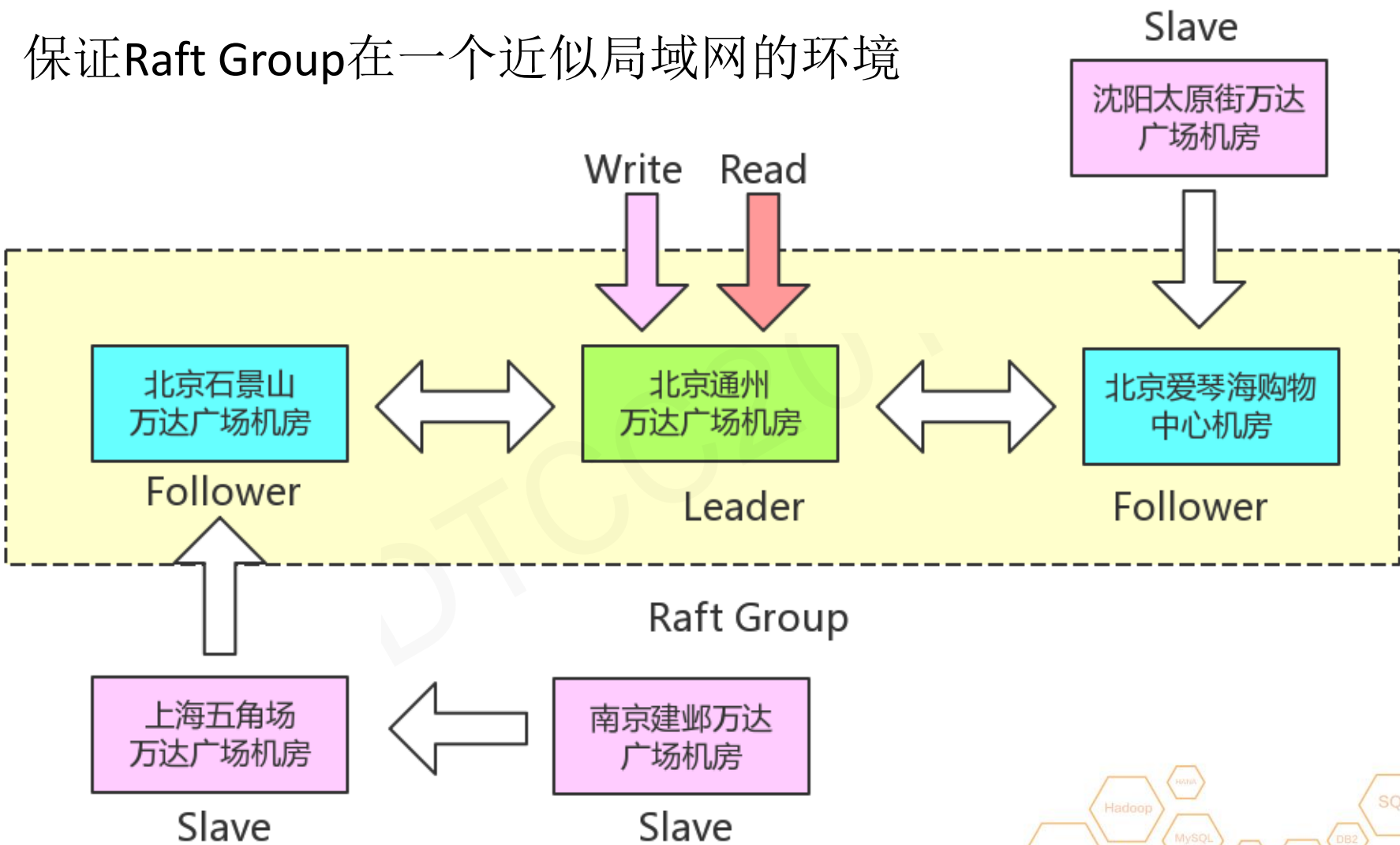


假设采用Paxos模型，我们的系统会有什么问题？

异地写，异地数据同步，丢包率与性能不达标

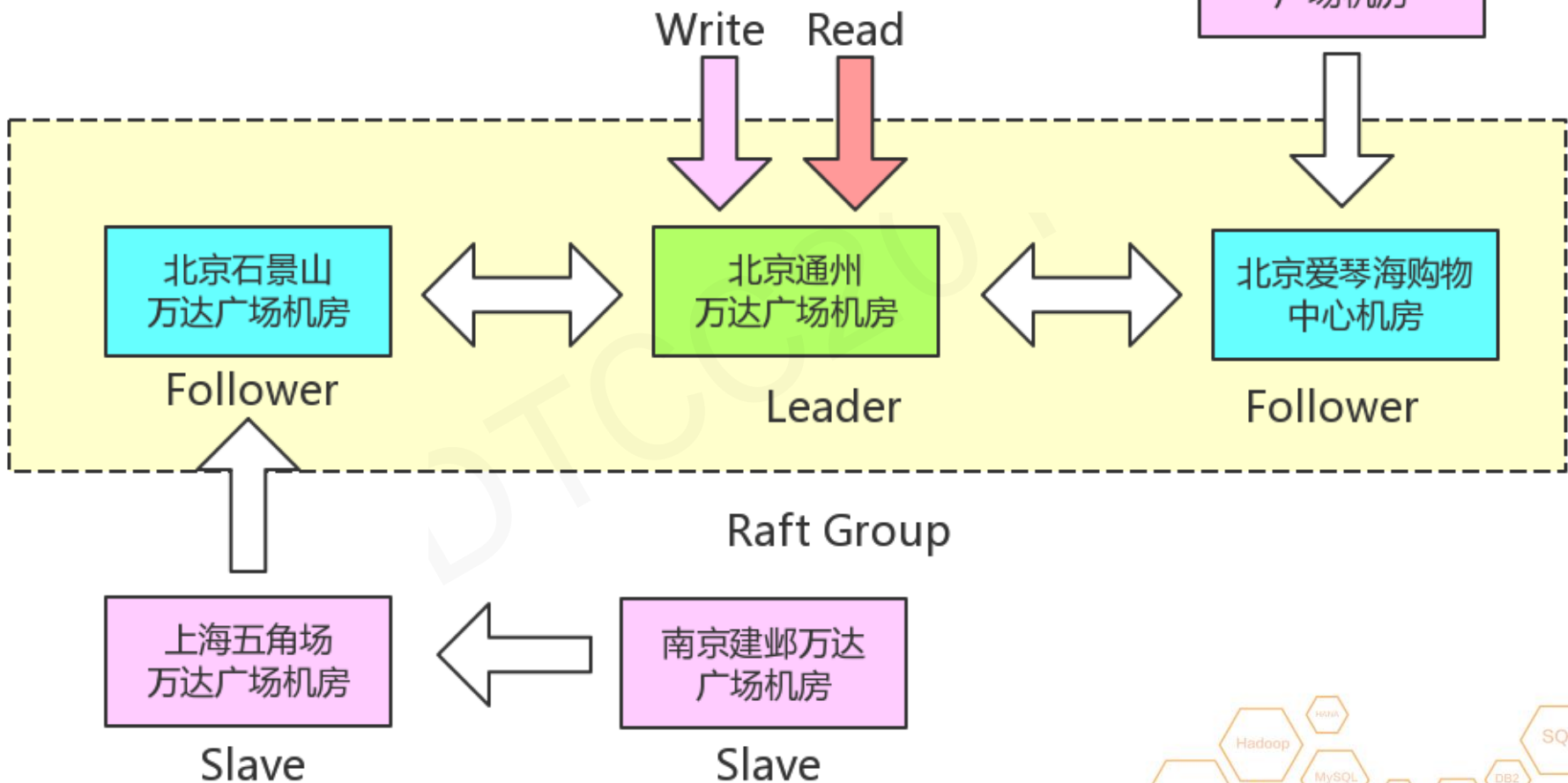


保证Raft Group在一个近似局域网的环境



Paxos类协议

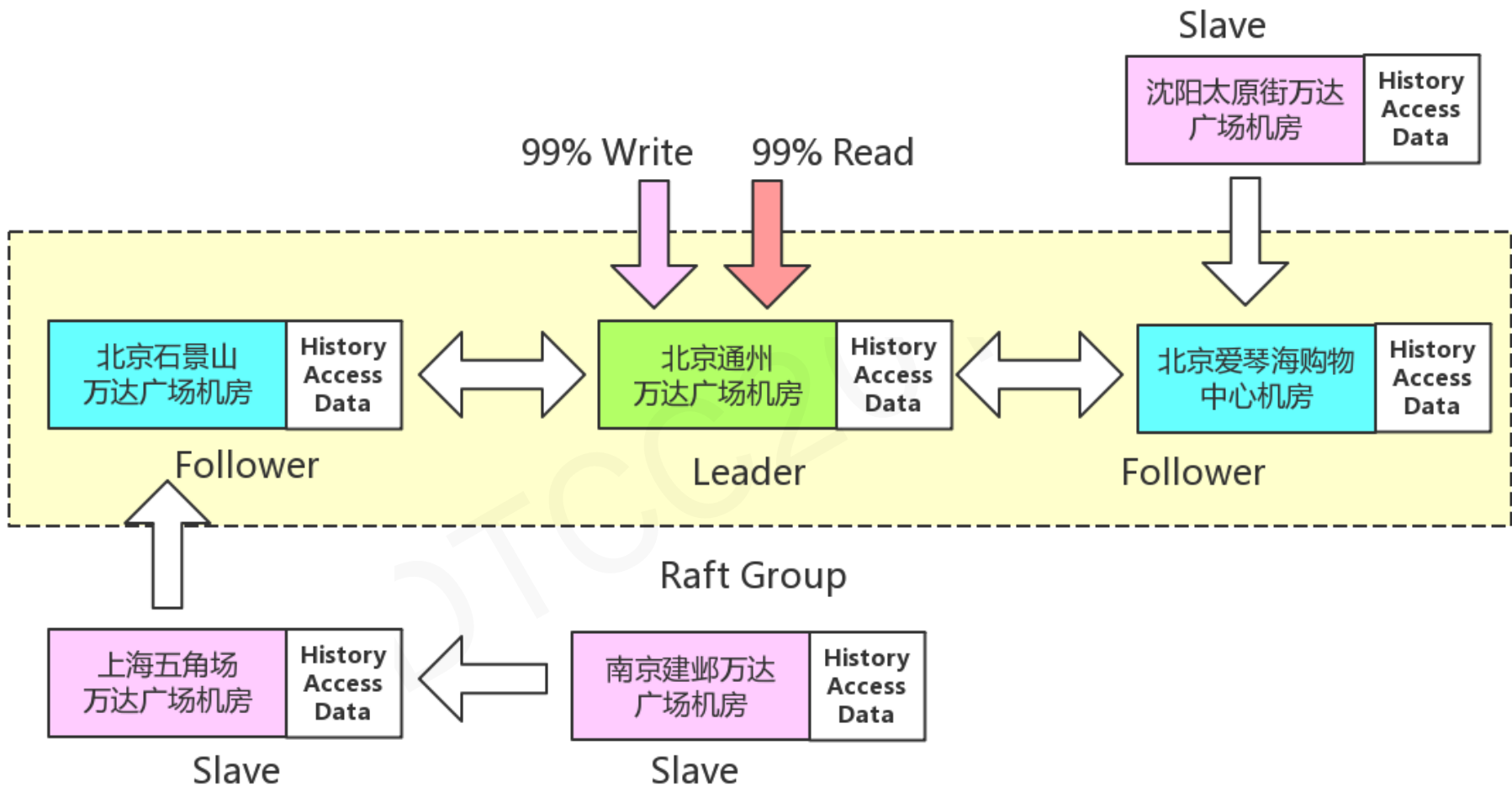
解决了RaftGroup数据同步的问题，
但仍然没有完全实现本地读写



抽样统计用户的对某条数据的读写地点，发现99%以上的读写都发生在数据所在的同一个城市

DTCC2018

Paxos类协议



1. 根据历史访问规律，对数据增加归属地信息
2. 99%以上的读写享受本地计算和本地存储的收益
3. 对数据的读操作，提供强一致和最终一致两种接口。强一致读归属地机房，最终一致读本地机房。
4. 系统统计用户的历史访问规律，对归属地信息做自适应的修改



优势： 99%以上的读写享受本地操作待遇

劣势： 受限于业务场景，不“普适”

DTCC2018



优势：99%以上的读写享受本地操作待遇

劣势：受限于业务场景，不“普适”

所谓的“优化”本质上是针对特定场景下的访问规律在时间和空间两个维度做自适应和更精确的调度

另一种思路：
VectorClock

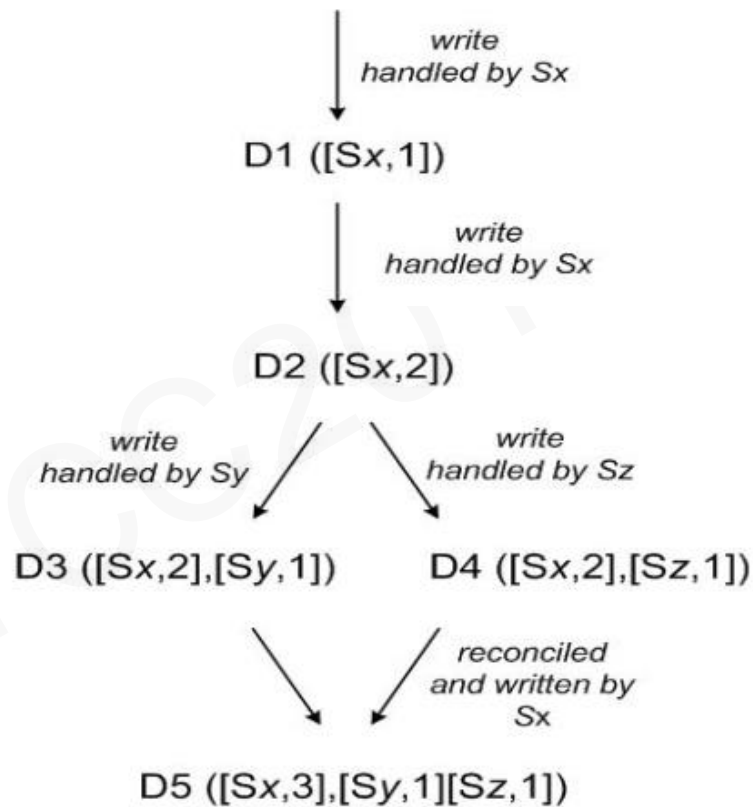


Figure 3: Version evolution of an object over time.

另一种思路：
VectorClock

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

Lamport, L. Time, clocks, and the ordering of events in a distributed system. ACM Communications, 21(7), pp. 558-565, 1978.

待探索：

引入机器学习与数据挖掘的算法，更精确的将与用户相关的数据自适应到合适的机房，甚至做到在用户物理上到达某个机房之前，做出预测，提前将数据调度过去。

这里的关键点在于用户线下的物理行为是有一定规律可循的，也是实体商业的一种特点。

是否靠谱，尚待研究和实践检验。。

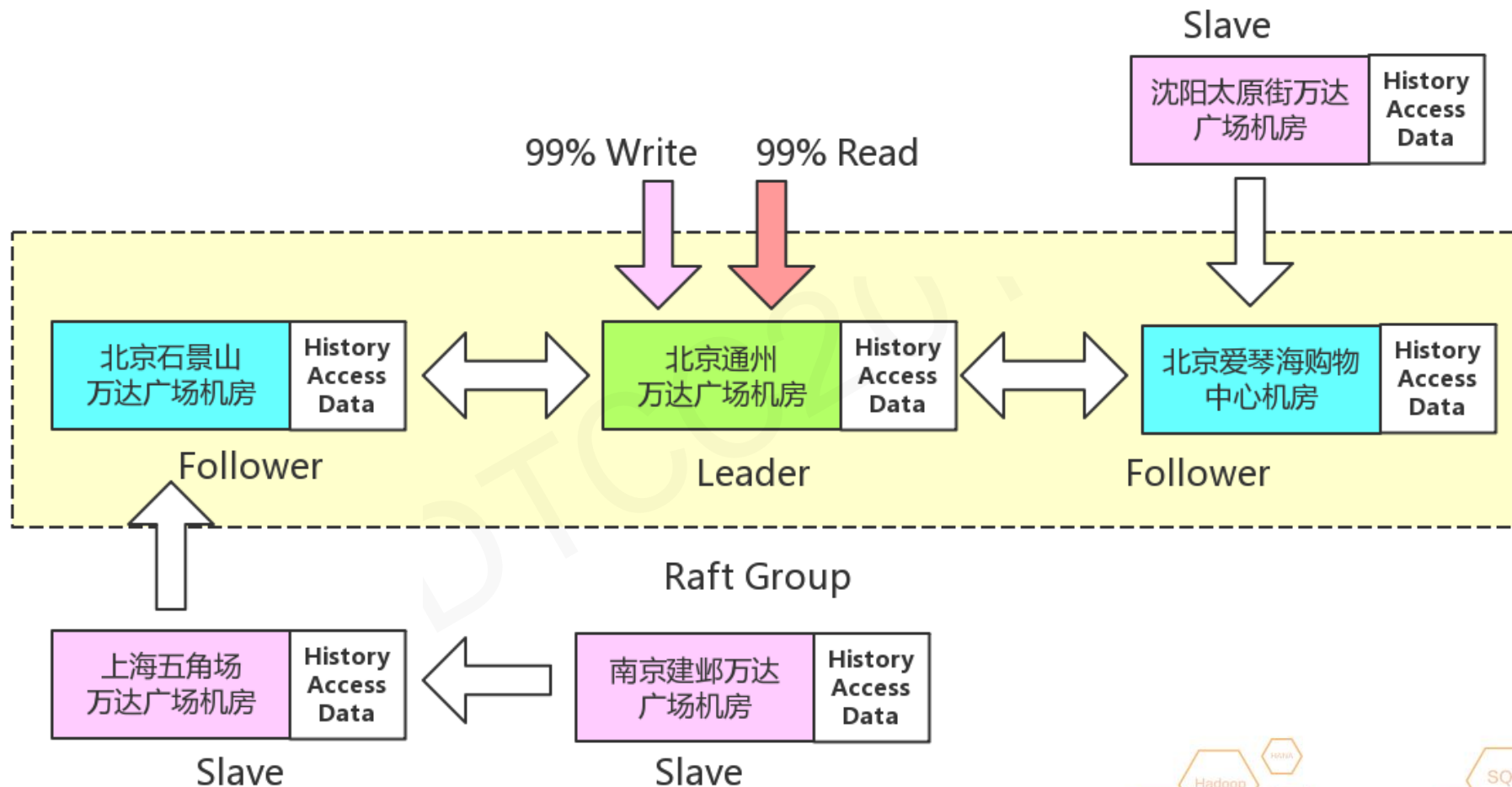
1. 提出问题

2. 分析问题

3. 解决问题

4. 经验总结

最终的选择



实现一个生产级别的Paxos库，并不是一件容易的事

DTCC2018

实现一个生产级别的Paxos库，并不是一件容易的事

Paxos Made Live - An Engineering Perspective

Tushar Chandra
Robert Griesemer
Joshua Redstone

June 26, 2007

Abstract

We describe our experience building a fault-tolerant data-base using the Paxos consensus algorithm. Despite the existing literature in the field, building such a database proved to be non-trivial. We describe selected algorithmic and engineering problems encountered, and the solutions we found for them. Our measurements indicate that we have built a competitive system.

实现一个生产级别的Paxos库，并不是一件容易的事

- While Paxos can be described with a page of **pseudo-code**, our complete implementation contains several **thousand lines of C++ code**. The blow-up is not due simply to the fact that we used C++ instead of pseudo **notation**, nor because our code style may have been **verbose**. Converting the algorithm into a practical, production-ready system involved implementing **many features and optimizations** – some published in the literature and **some not**.
- The fault-tolerant algorithms community is **accustomed** to proving short algorithms (one page of pseudo code) correct. This approach does not scale to a system with thousands of lines of code. To gain confidence in the “correctness” of a real system, different methods had to be used.
- Fault-tolerant algorithms tolerate **a limited set of carefully selected faults**. However, the real world **exposes** software to a wide variety of failure modes, including errors in the algorithm, bugs in its



解决问题

<https://github.com/Tencent/phxpaxos>



Features

Business

Explore

Marketplace

Pricing

This repository

Search

Sign in or Sign up

Tencent / phxpaxos

Watch

264

★ Star

1,804

🍴 Fork

531

<> Code

🔔 Issues 25

🔗 Pull requests 2

📁 Projects 0

📖 Wiki

📊 Insights

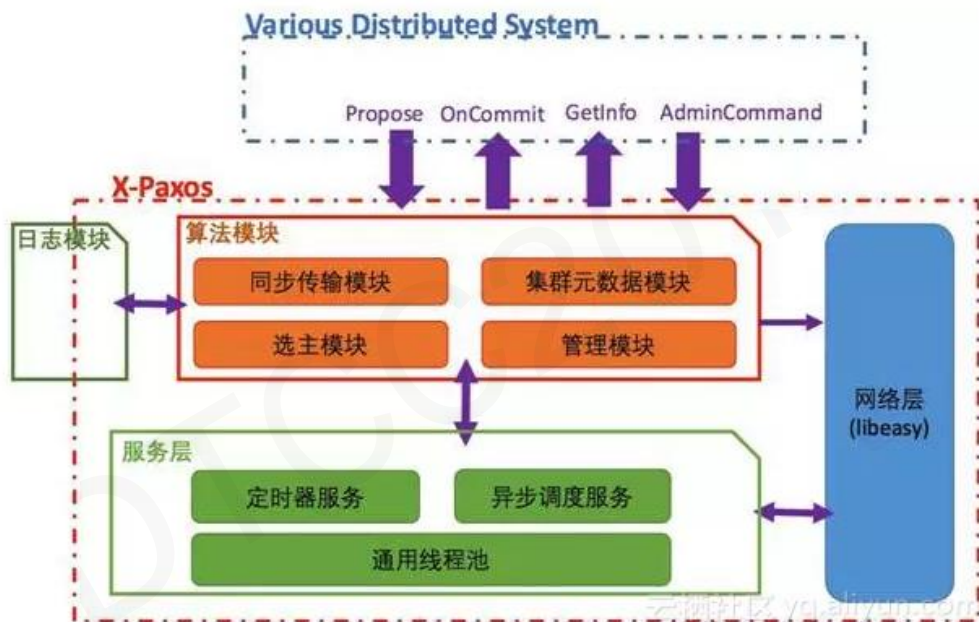
DTCC
2018

数领先机 智赢未来 (9)

IT168.com

ChinaUnix

ITPUB



X-Paxos的整体架构如上图所示，主要可分为网络层、服务层、算法模块、日志模块4个部分

解决问题

<https://github.com/logcabin/logcabin>



Features

Business

Explore

Marketplace

Pricing

This repository

Search

Sign in or Sign up

 logcabin / logcabin

Watch

86

★ Star

938

Fork

150

<> Code

Issues 55

Pull requests 3

Projects 0

Insights

DTCC
2018

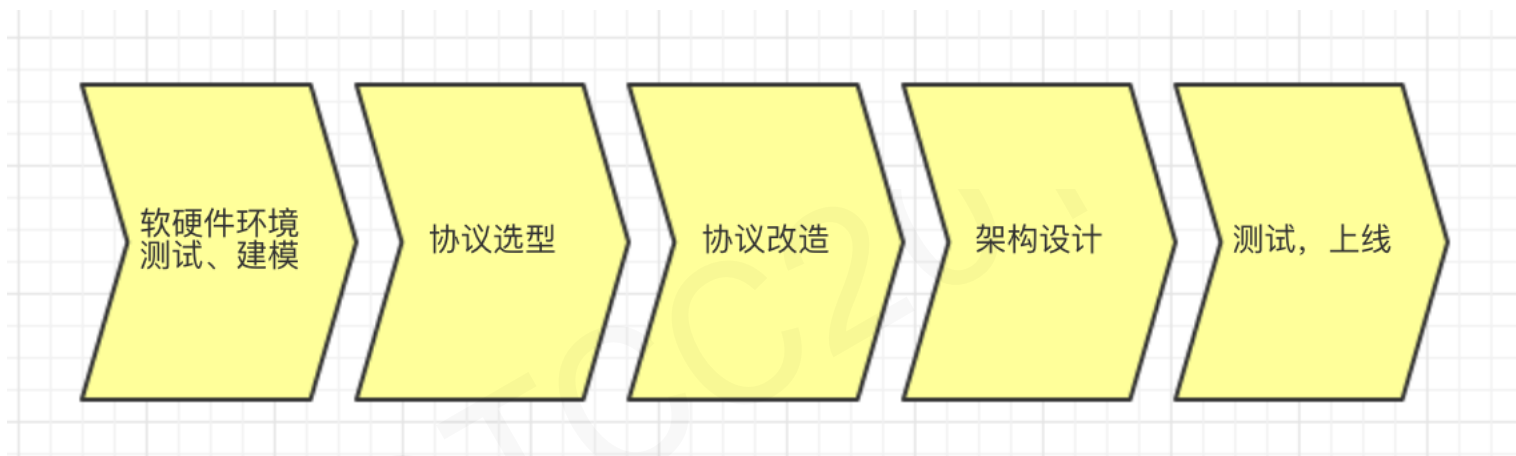
数领先机 智赢未来 (9)

IT168.com

ChinaUnix

ITPUB

系统设计与实践的流程



1. 提出问题

2. 分析问题

3. 解决问题

4. 经验总结

1.理论决定上限，经验决定下限

DTCC2018

- 1.理论决定上限，经验决定下限
- 2.架构师的抽象思维能力很重要

DTCC2018



- 1.理论决定上限，经验决定下限
- 2.架构师的抽象思维能力很重要
- 3.天马行空的思考，脚踏实地的实践

- 1.理论决定上限，经验决定下限
- 2.架构师的抽象思维能力很重要
- 3.天马行空的思考，脚踏实地的实践
- 4.实践！ 实践！ 实践！

谢谢大家

DTCC
2018

数领先机 智赢未来 (9)

IT168.com

ChinaUnix

ITPUB

THANKS





讲师申请

联系电话（微信号）：18612470168

关注“ITPUB”更多
技术干货等你来拿~

与百度外卖、京东、魅族等先后合作系列分享活动



让学习更简单

微学堂是以ChinaUnix、ITPUB所组建的微信群为载体，定期邀请嘉宾对热点话题、技术难题、新产品发布等进行移动端的在线直播活动。

截至目前，累计举办活动期数60+，参与人次40000+。

ITPUB学院

ITPUB学院是盛拓传媒IT168企业事业部（ITPUB）旗下
企业级在线学习咨询平台
历经18年技术社区平台发展
汇聚5000万技术用户
紧随企业一线IT技术需求
打造全方式技术培训与技术咨询服务
提供包括企业应用方案培训咨询（包括企业内训）
个人实战技能培训（包括认证培训）
在内的全方位IT技术培训咨询服务

ITPUB学院讲师均来自于企业
一些工程师、架构师、技术经理和CTO
大会演讲专家1800+
社区版主和博客专家500+

培训特色

无限次免费播放
随时随地在线观看
碎片化时间集中学习
聚焦知识点详细解读
讲师在线答疑
强大的技术人脉圈

八大课程体系

基础架构设计与建设
大数据平台
应用架构设计与开发
系统运维与数据库
传统企业数字化转型
人工智能
区块链
移动开发与SEO



联系我们

联系人：黄老师
电话：010-59127187
邮箱：edu@itpub.net
网址：edu.itpub.net
培训微信号：18500940168