

分布式OLTP DB

上节回顾

字幕课程

LAST CLASS

System Architectures

→ Shared-Everything, Shared-Disk, Shared-Nothing

Partitioning/Sharding

→ Hash, Range, Round Robin

Transaction Coordination

→ Centralized vs. Decentralized

OLTP VS . OLAP

| 执行时间30/50ms以上的事务被认定是长事务

英字幕课程

OLTP VS. OLAP

On-line Transaction Processing (OLTP):

- Short-lived read/write txns.
- Small footprint.
- Repetitive operations.

On-line Analytical Processing (OLAP):

- Long-running, read-only queries.
- Complex joins.
- Exploratory queries.

今天要实现的大致框架

中英字幕课程

DECENTRALIZED COORDINATOR

Application Server

Commit Request

Primary Node

Partitions

P1

P2

P3

P4

Safe to commit?

1-DB (Fall 2023)

这就是我们今天要实现的大致框架。

So that's the big picture of what we're trying to do today.

目标：多个物理节点被视作一个逻辑上的DBMS

中英字幕课程

OBSERVATION

Recall that our goal is to have multiple physical nodes appear as a single logical DBMS.

We have not discussed how to ensure that all nodes agree to commit a txn and then to make sure it does commit if the DBMS decides it should.

- What happens if a node fails?
- What happens if messages show up late?
- What happens if the system does not wait for every node to agree to commit?

如果在系统崩溃或发生故障的情况下，当系统重新启动时，如果我们曾向外界声明某个事务已提交，我们必须确保这一

And if there's a crash or there's a failure, that when the system comes back up, that

环境假设：无需保证拜占庭协议

IMPORTANT ASSUMPTION

We will assume that all nodes in a distributed DBMS are well-behaved and under the same administrative domain.

→ If we tell a node to commit a txn, then it will commit the txn (if there is not a failure).

If you do not trust the other nodes in a distributed DBMS, then you need to use a Byzantine Fault Tolerant protocol for txns (blockchain).

→ This is stupid. The real world doesn't work this way.

这意味着我们并非处于一个奇异且不可信的环境中，在这个环境中，我们的分布式数据库中存在一些我们无法控制且租用的

Agenda

TODAY'S AGENDA

Replication

Atomic Commit Protocols

Consistency Issues (CAP / PACELC)

Google Spanner

我们将逐一探讨这些关键点，它们是你必须了解的存在事实，以及当你拥有一个分布式数据库系统将面临的挑战。

复制

可以对只读查询进行负载分流

REPLICATION

The DBMS can replicate a database across redundant nodes to increase availability.

- Partitioned vs. Non-Partitioned
- Shared-Nothing vs. Shared-Disk

Design Decisions:

- Replica Configuration
- Propagation Scheme
- Propagation Timing
- Update Method

所以分区意味着，如果我将数据库分割成不相交的集合，我仍然希望拥有这些不相交子集的多个副本。

have multiple copies of those disjoint subsets.

如果数据库未进行分区，这在大多数数据库系统中是常见情况，那么我希望能够利用副本进行负载分流，例如处理只读

1. 复制的设置

REPLICA CONFIGURATIONS

Approach #1: Primary-Replica

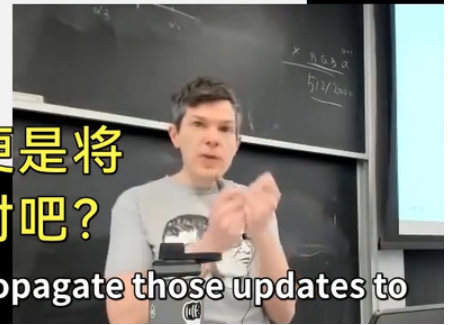
- All updates go to a designated primary for each object.
- The primary propagates updates to its replicas without an atomic commit protocol.
- Read-only txns may be allowed to access replicas.
- If the primary goes down, then hold an election to select a new primary.

Approach #2: Multi-Primary

- Txns can update data objects at any replica.
- Replicas must synchronize with each other using an atomic commit protocol.

并且，这一主要节点的责任便是将这些更新传播至任何副本，对吧？

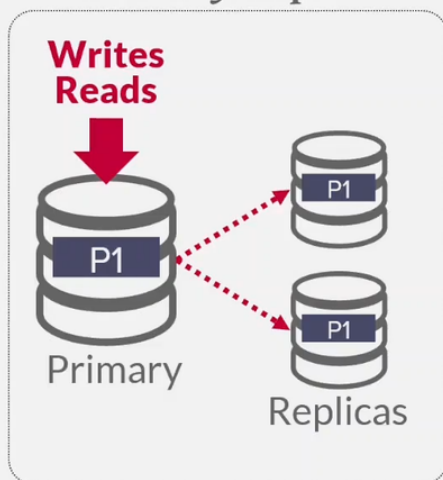
And it's going to be that primary's responsibility to then propagate those updates to



2. 单一主备份

REPLICA CONFIGURATIONS

Primary-Replica



这通常只是发送预写式日志
(write ahead log)。

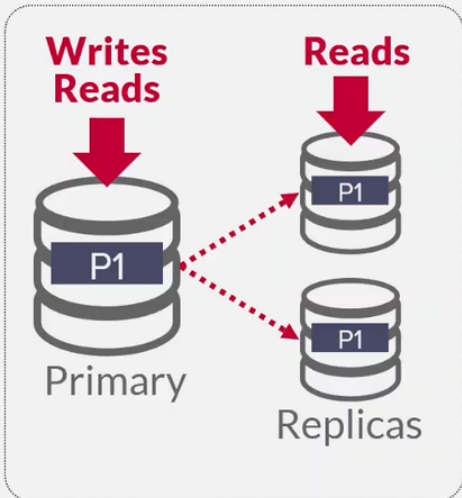
And this is typically just sending the write ahead log.



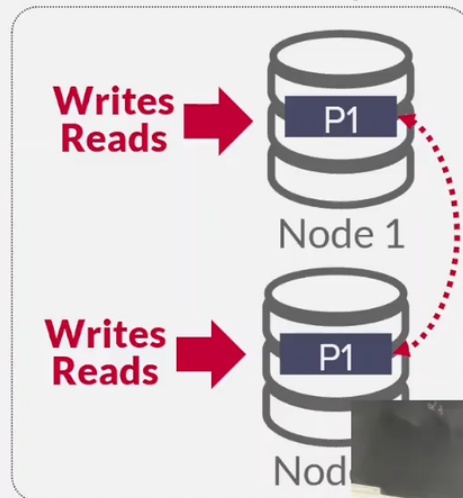
- 读写都发给主，写回传播到副本（只是WAL）
- 中间件可以将只读操作路由到副本上查询

REPLICA CONFIGURATIONS

Primary-Replica

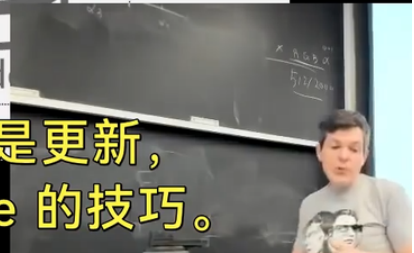


Multi-Primary



DB (Fall 2023)

因为在执行操作时，我们进行的是更新，而你无法在其中应用浏览器 cookie 的技巧。



对于数据库给定对象的所有副本，我们可以容忍多少节点发生故障

K是可用副本数

K-SAFETY

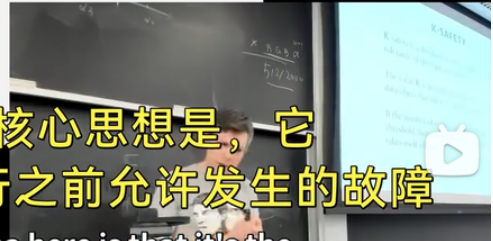
K-safety is a threshold for determining the fault tolerance of the replicated database.

The value *K* represents the number of replicas per data object that must always be available.

If the number of replicas goes below this threshold, then the DBMS halts execution and takes itself offline.

J-DB (Fall 2023)

我认为这主要是一个数据库术语，但其核心思想是，它规定了分布式数据库系统在决定不再继续进行之前允许发生的故障



传播方案

PROPAGATION SCHEME

When a txn commits on a replicated database, the DBMS decides whether it must wait for that txn's changes to propagate to other nodes before it can send the acknowledgement to application.

Propagation levels:

- Synchronous (*Strong Consistency*)
- Asynchronous (*Eventual Consistency*)

这就是我们将如何决定何时以及如何将主数据库的更改传播到副本数据库的方式。

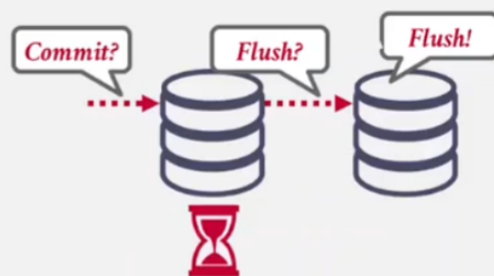
So this is how we're going to decide when and how we will propagate the changes from

同步 (强一致性)

需要等待副本成功写入后才进行事务提交

Approach #1: Synchronous

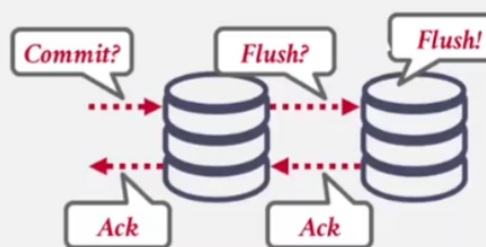
→ The primary sends updates to replicas and then waits for them to acknowledge that they fully applied (i.e., logged) the changes.



PROPAGATION SCHEME

Approach #1: Synchronous

→ The primary sends updates to replicas and then waits for them to acknowledge that they fully applied (i.e., logged) the changes.

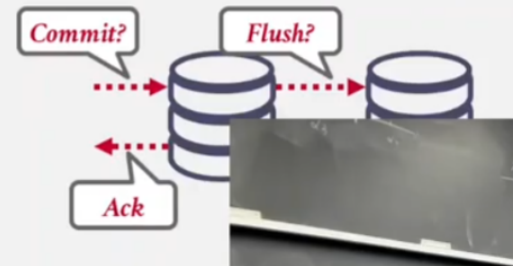


异步 (最终一致性)

副本接收到直接返回ACK

Approach #2: Asynchronous

→ The primary immediately returns the acknowledgement to the client without waiting for replicas to apply the changes.



CMU-DB

Propagation Timing

连续传播：一旦产生一个更新，立即传播log message

提交传播：等到事务提交时才传播log message

GPT中英字幕课程

16

PROPAGATION TIMING

Approach #1: Continuous

- The DBMS sends log messages immediately as it generates them.
- Also need to send a commit/abort message.

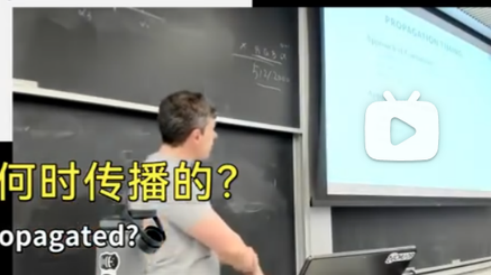
Approach #2: On Commit

- The DBMS only sends the log messages for a txn to the replicas once the txn is commits.
- Do not waste time sending log records for aborted txns.
- Assumes that a txn's log records fits entirely in memory.

CMU-DB
15-445/6-65 (Fall 2023)

接下来要探讨的是，变更实际上是在何时传播的？

The next is when are the changes actually propagated?



Active-Active：每个事务在每个副本独立执行，破坏只读查询的效率性

Active-Passive：每个事务只在本地修改并传播到其他副本

ACTIVE VS. PASSIVE

Approach #1: Active-Active

- A txn executes at each replica independently.
- Need to check at the end whether the txn ends up with the same result at each replica.

Approach #2: Active-Passive

- Each txn executes at a single location and propagates the changes to the replica.
- Can either do physical or logical replication.
- Not the same as Primary-Replica vs. Multi-Primary

所以 ActivePath 就是我迄今为止所描述的那种机制，其中存在一个主节点，查询请求发往该处，执行更新操作，随后这些

So ActivePath is what I've been sort of describing so far, where there's some

Atomic Commit Protocol

ATOMIC COMMIT PROTOCOL

Coordinating the commit order of txns across nodes in a distributed DBMS.

- Commit Order = State Machine
- It does not matter whether the database's contents are replicated or partitioned.

Examples:

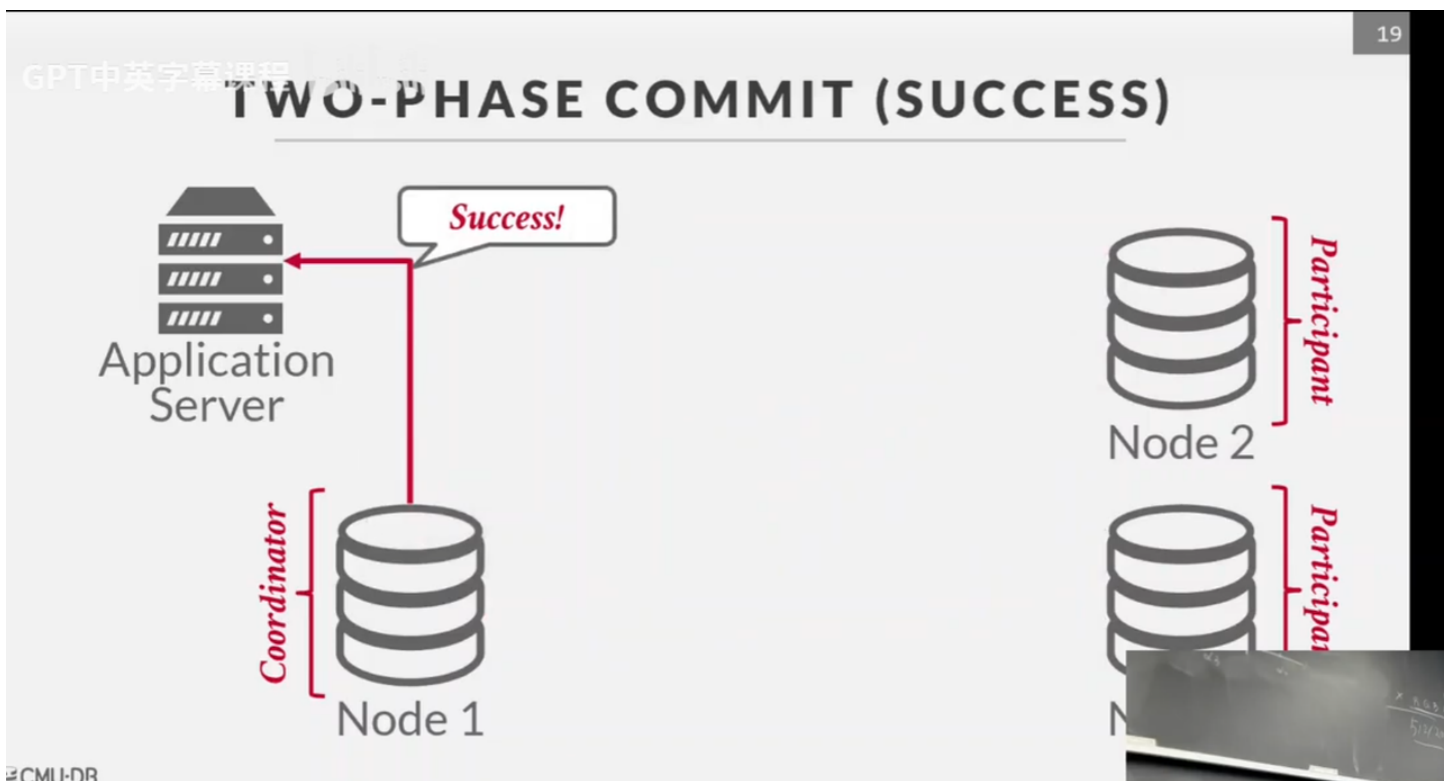
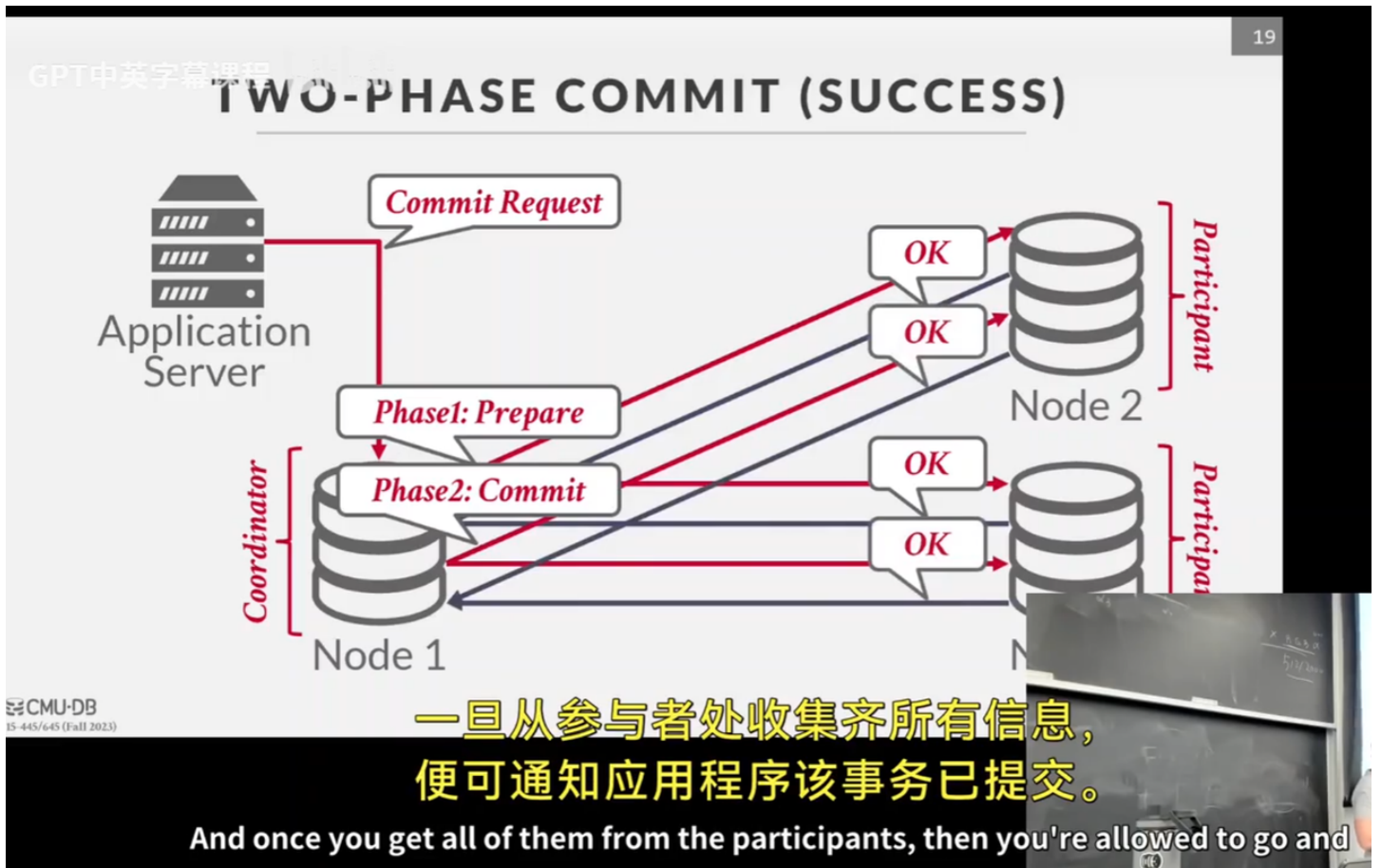
- Two-Phase Commit (1970s)
- Three-Phase Commit (1983)
- Viewstamped Replication (1988)
- Paxos (1989)
- ZAB (2008?)
- Raft (2013)

Two-Phase Commit

我们必须等待所有参与者返回确认信息后，才能进行下一阶段

任何一个节点都可能导致整个系统崩溃

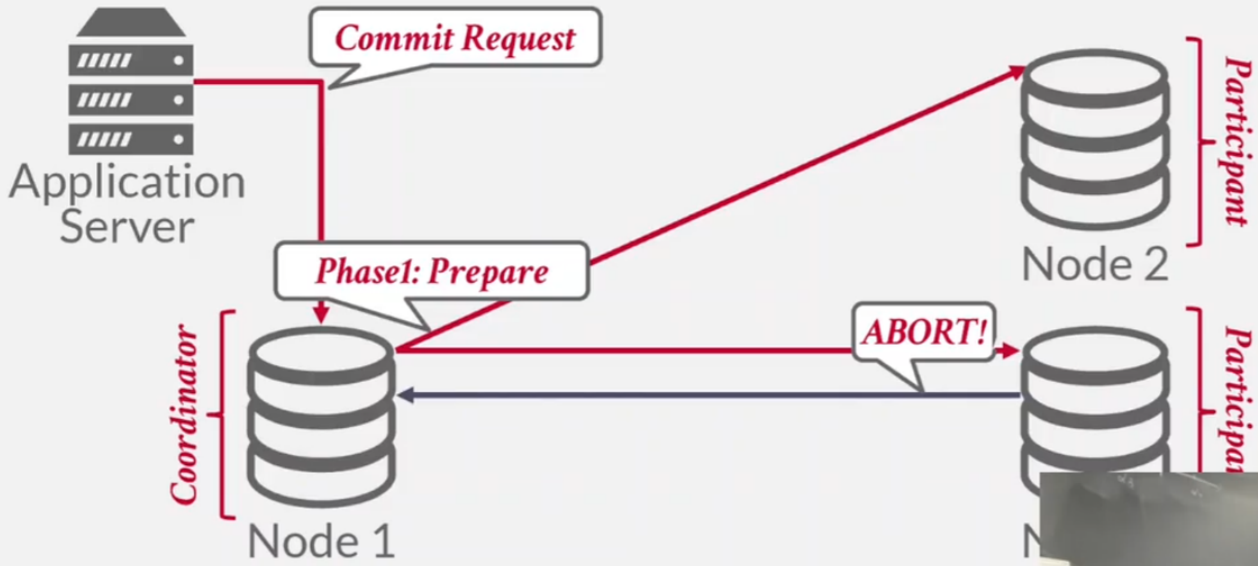
success



abort

当收到abort，直接向application server回复aborted

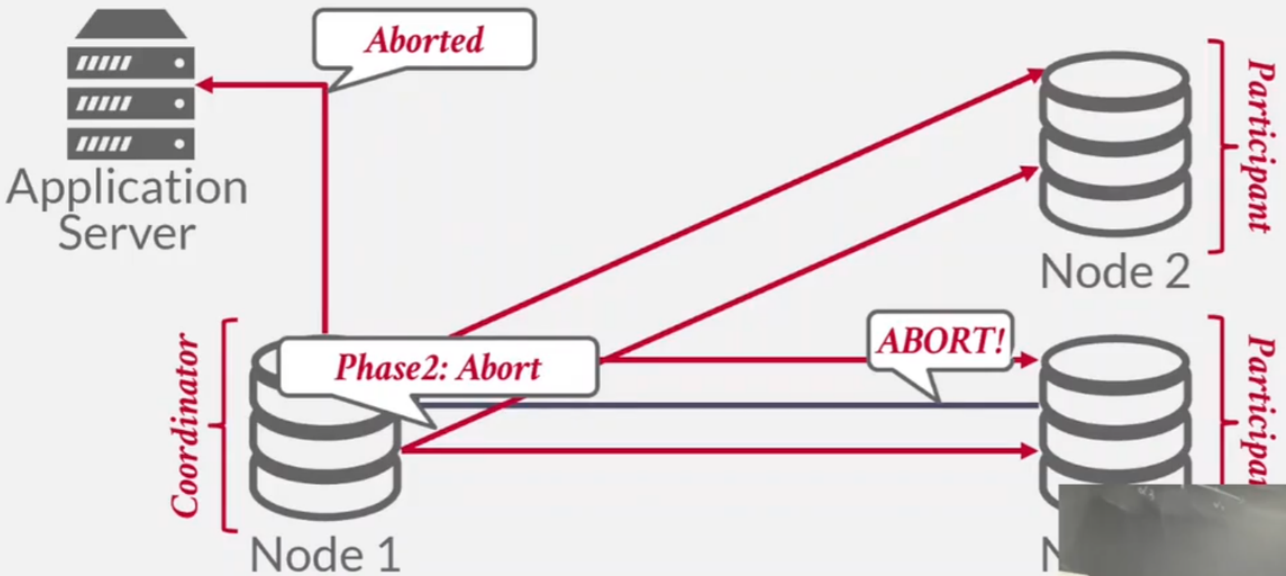
TWO-PHASE COMMIT (ABORT)



CMU-DB 5-445/6-45 (Fall 2023)

如果在网络中有一个参与者表示希望中止此次事务，那么我们可以立即告知外界，我们的事务已经中止。
 immediately we can tell the outside world that our transaction has aborted.

TWO-PHASE COMMIT (ABORT)



CMU-DB 5-445/6-45 (Fall 2023)

即便如此，即使某个节点可能表示过：“哦，是的，我确实想要提交这个事务。”
 So even though one node might have said, oh, yeah, I really want to commit

recovery

TWO-PHASE COMMIT

Each node records the inbound/outbound messages and outcome of each phase in a non-volatile storage log.

On recovery, examine the log for 2PC messages:

- If local txn in prepared state, contact coordinator.
- If local txn not in prepared, abort it.
- If local txn was committing and node is the coordinator, send **COMMIT** message to nodes.

Failures

不可用：无法接受任何新的查询，也无法接受任何写入

TWO-PHASE COMMIT FAILURES

What happens if coordinator crashes?

- Participants must decide what to do after a timeout.
- System is not available during this time.

What happens if participant crashes?

- Coordinator assumes that it responded with an abort if it has not sent an acknowledgement yet.
- Again, nodes use a timeout to determine whether a participant is dead.

2PC Optimizations

一般采用early ack after prepare

2PC OPTIMIZATIONS

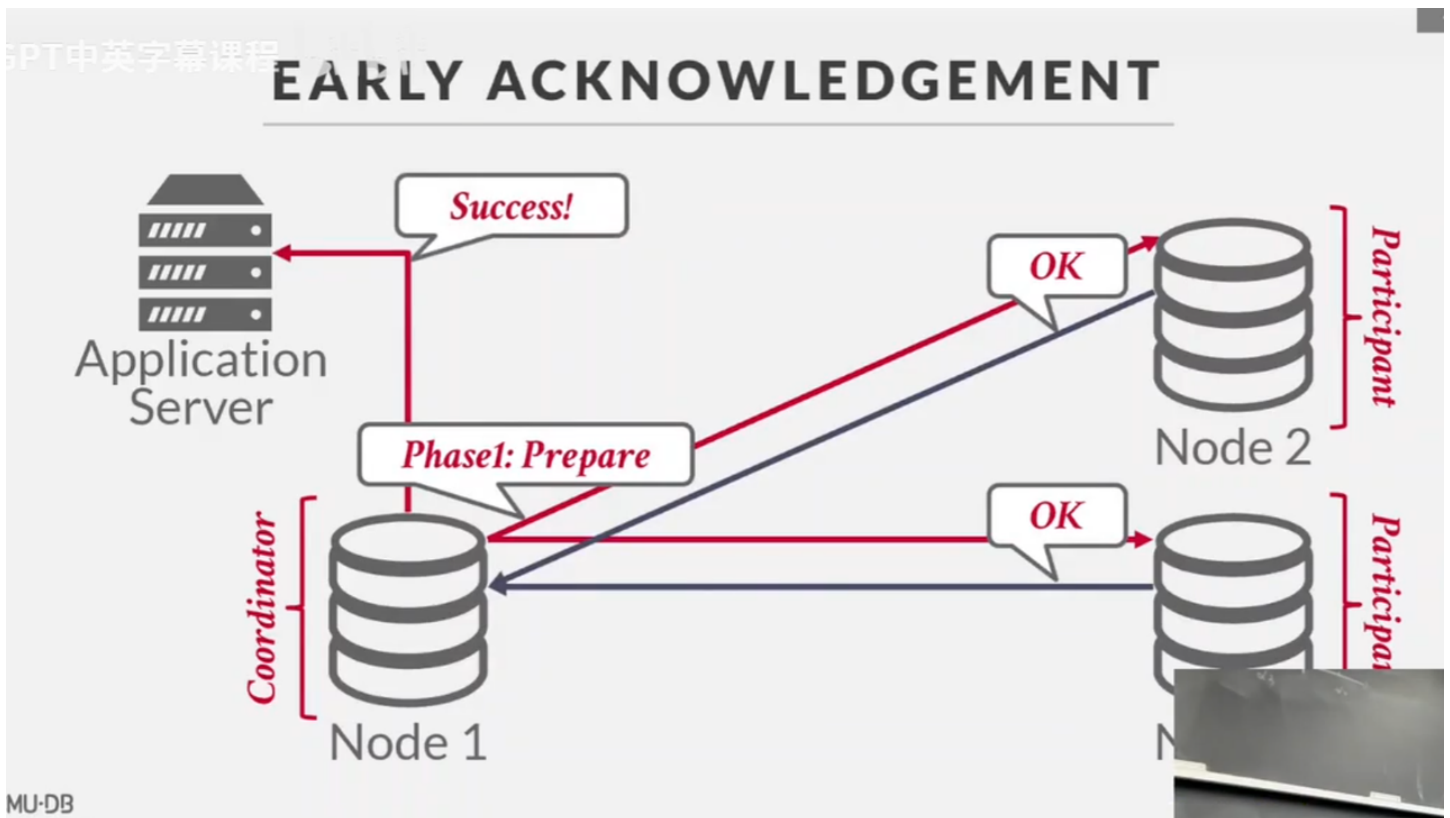
Early Prepare Voting (*Rare*)

→ If you send a query to a remote node that you know will be the last one you execute there, then that node will also return their vote for the prepare phase with the query result.

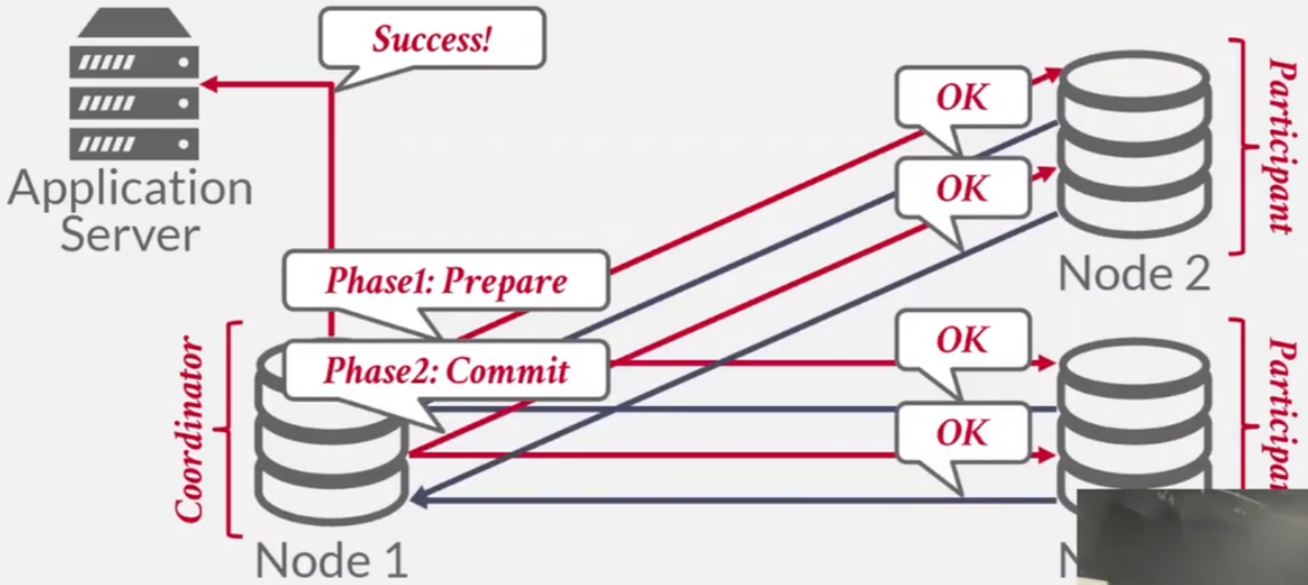
Early Ack After Prepare (*Common*)

→ If all nodes vote to commit a txn, the coordinator can send the client an acknowledgement that their txn was successful before the commit phase finishes.

Early ACK After Prepare



EARLY ACKNOWLEDGEMENT



同样，所有这些都将被记录到日志中，并非所有人都会这么做。

PAXOS

多数投票者同意即可成功；

少数派成员被认为失败或者故障，通过重放日志来恢复正确的状态

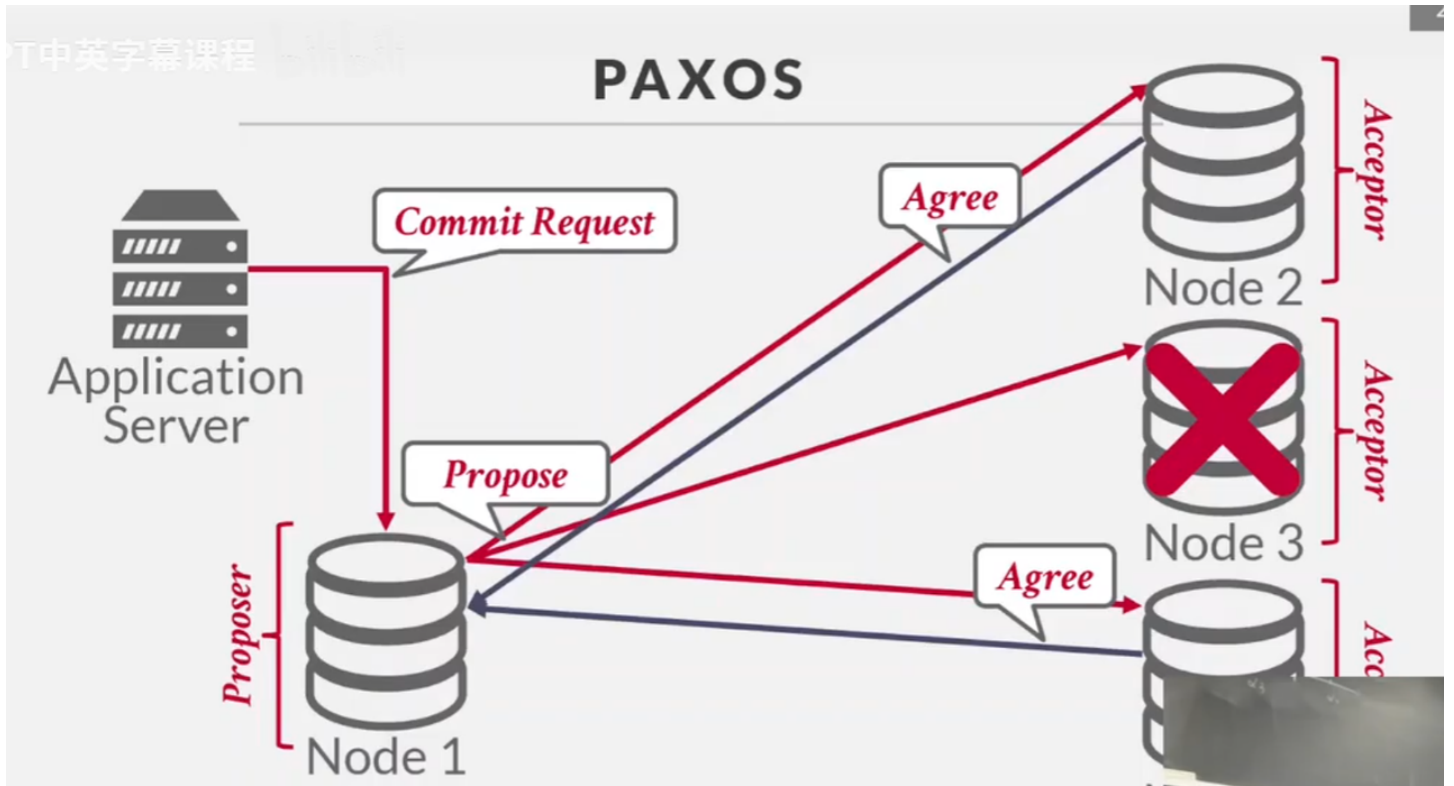
PAXOS

Consensus protocol where a coordinator proposes an outcome (e.g., commit or abort) and then the participants vote on whether that outcome should succeed.

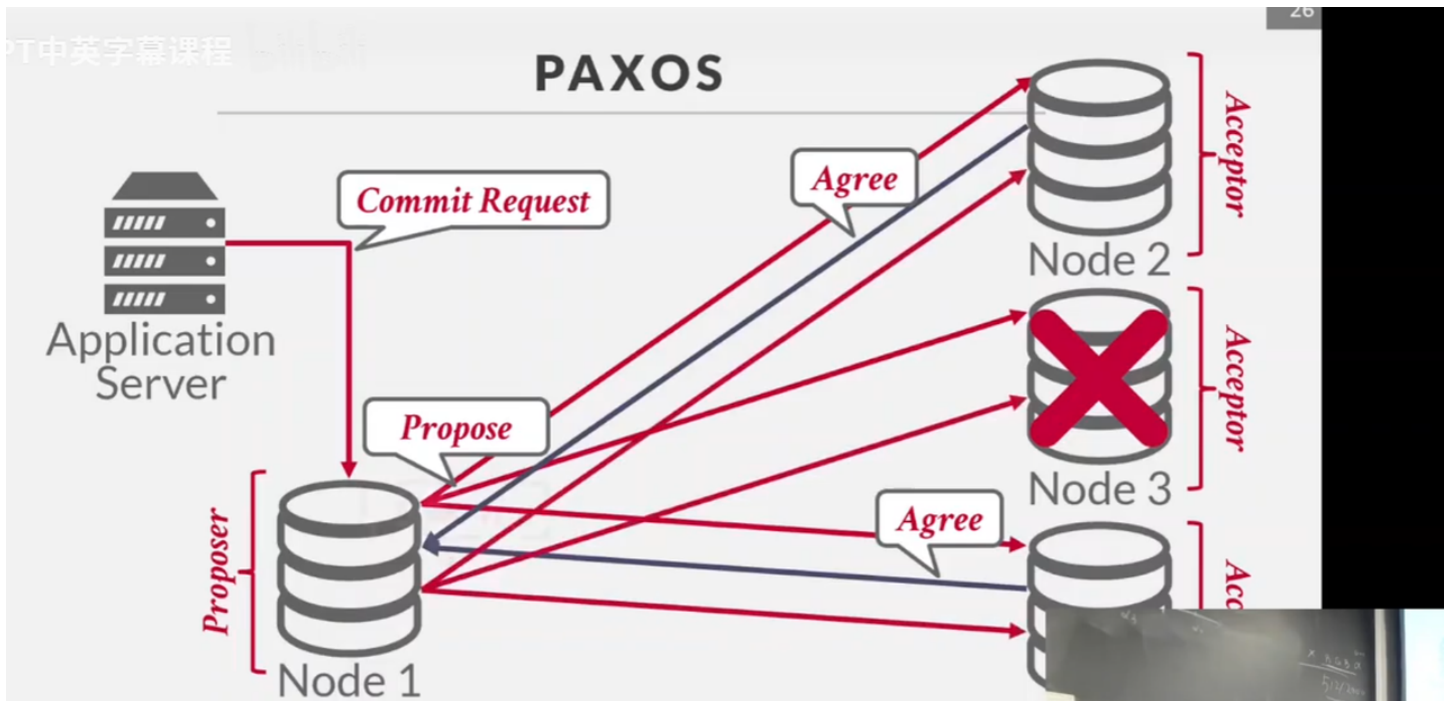
Does not block if a majority of participants are available and has provably minimal message delays in the best case.



然而，与其让所有参与者返回并确认该事务被允许提交，你需要的是多数同意。



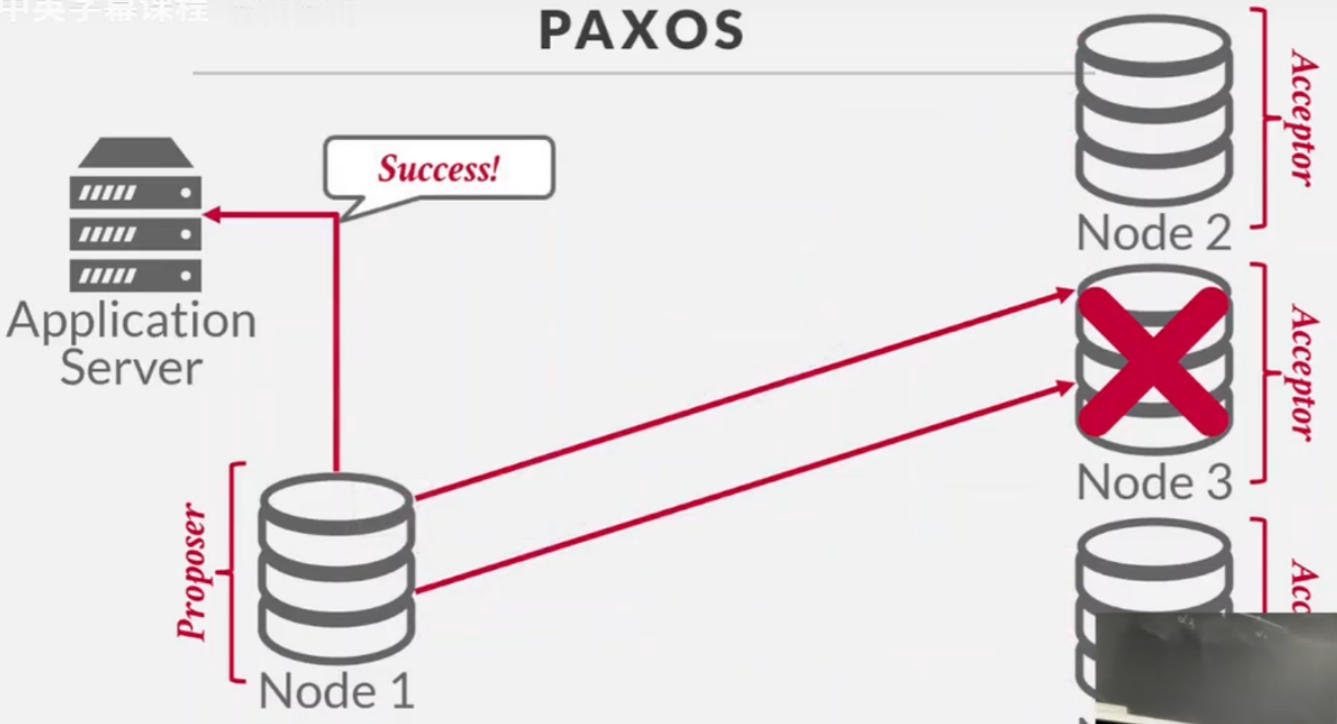
由于我们得到了三个节点中的两个同意我们可以提交此交易，该交易被允许进行，于是我们执行提交阶段，之后该交易便



由于我们得到了三个节点中的两个同意我们可以提交此交易，该交易被允许进行，于是我们执行提交阶段，之后该交易便

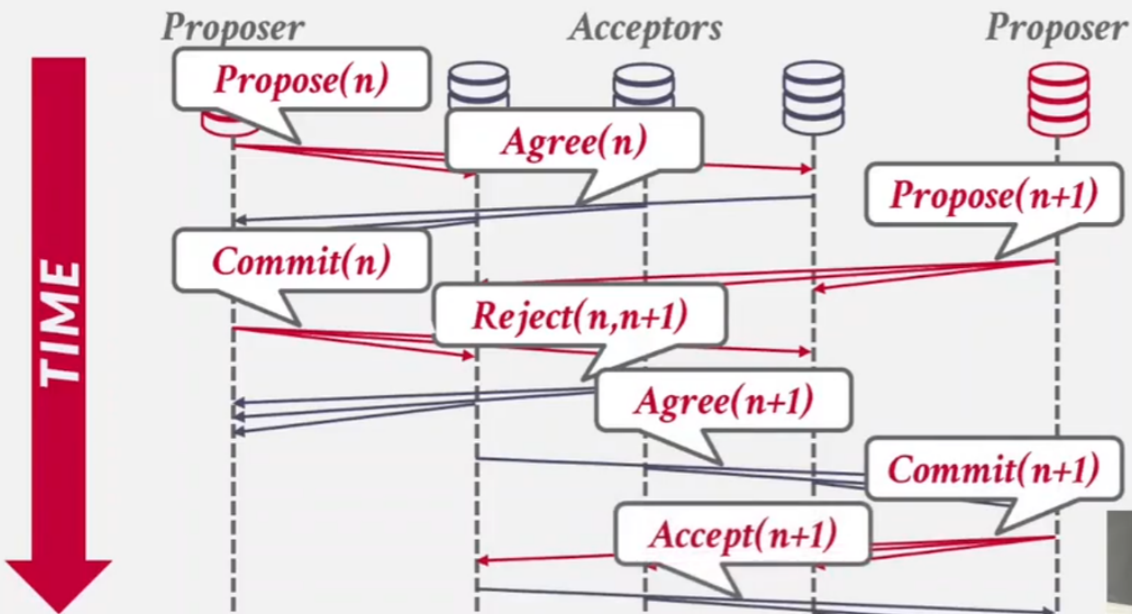
Node 3重新上线后，可以重放日志获取到新的变更

PAXOS



Time

PAXOS



所以，现在他们可以继续同意提交 n 加一，完成这个操作，然后这里的另一位，他可以重新提交他的请求以

Multi-Paxos

如果系统选举出一个单一领导者，在一段时间内监督提出变更，则可以跳过 Propose 阶段。

当出现故障时退回到完整的 Paxos 协议。

可以设置自动超时时间，超时后会自动重新进行一次领导者选举

GPT中英字幕课程

MULTI-PAXOS

If the system elects a single leader that oversees proposing changes for some period, then it can skip the **Propose** phase.

→ Fall back to full Paxos whenever there is a failure.

The system periodically renews the leader (known as a *lease*) using another Paxos round.

→ Nodes must exchange log entries during leader election to make sure that everyone is up-to-date.

CMU-DB
15-445/645 (Fall 2023)

然后，如果在任何时候发生了故障，无论是领导者节点还是其他某个节点宕机，你只需重新进行领导者选举

Multi paxos有领导者，而paxos没有领导者，任何人都可以是领导者、参与者；但是彼此的提交会相互覆盖，需要采取退让策略

字幕课程

2PC VS. PAXOS VS. RAFT

Two-Phase Commit

→ Blocks if coordinator fails after the prepare message is sent, until coordinator recovers.

Paxos

→ Non-blocking if a majority participants are alive, provided there is a sufficiently long period without further failures.

Raft:

→ Similar to Paxos but with fewer node types.

→ Only nodes with most up-to-date log can become leader

CAP Theorem

分布式数据库只能从CAP中选择两个条件满足

- C是一致性
- A是Always Available
- N是Network Partition Tolerant

PT中英字幕课程

CAP THEOREM

Proposed in the late 1990s that is impossible for a distributed database to always be:

- Consistent
- Always Available
- Network Partition Tolerant

Extended in 2010 (PACELC) to include consistency vs. latency trade-offs:

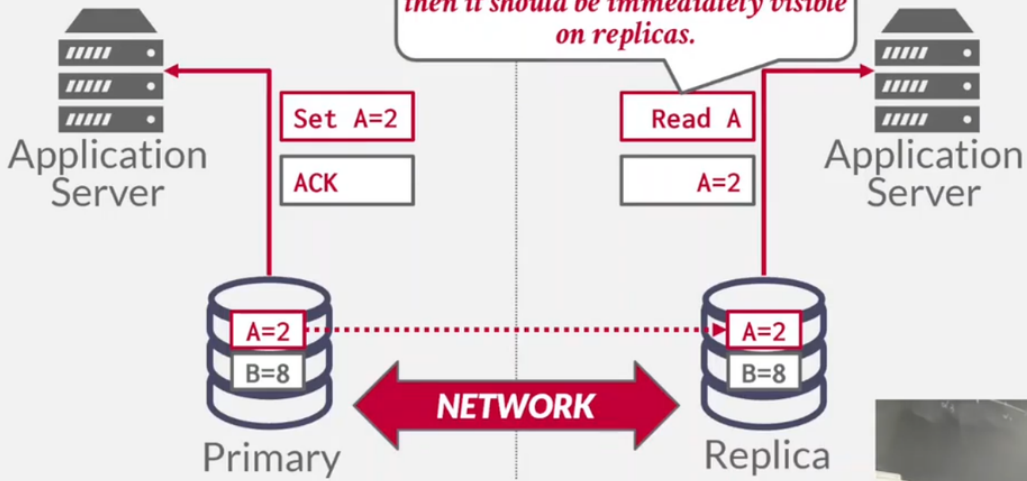
- Partition Tolerant
- Always Available
- Consistent
- Else, choose during normal operations
- Latency
- Consistency

U-DB
5 (Fall 2023)

Consistency

CONSISTENCY

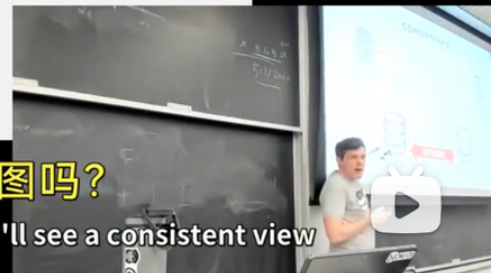
If Primary says the txn committed, then it should be immediately visible on replicas.



CMU-DB
IS-445/645 (Fall 2023)

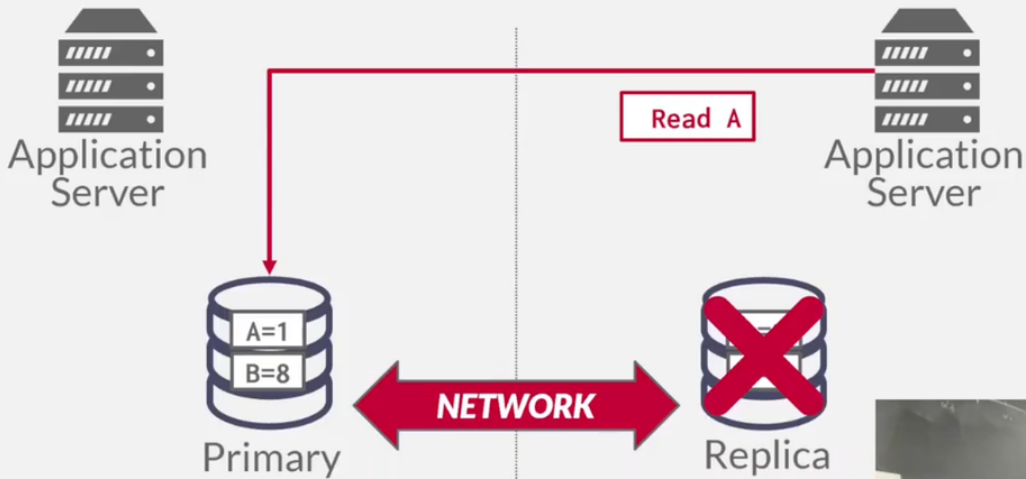
如果我同时更新 A 和 B, 我能保证看到数据库的一致视图吗?

If I update A and B at the same time, can I guarantee that I'll see a consistent view



Availability

AVAILABILITY



CMU-DB
IS-445/645 (Fall 2023)

内容。就像我假设网络没有被切断一样。好的，无论多少次多少节点宕机，分区容错性表明如果网络被切断，现在这
read the database over there Like I'm you know assuming the network it hasn't been

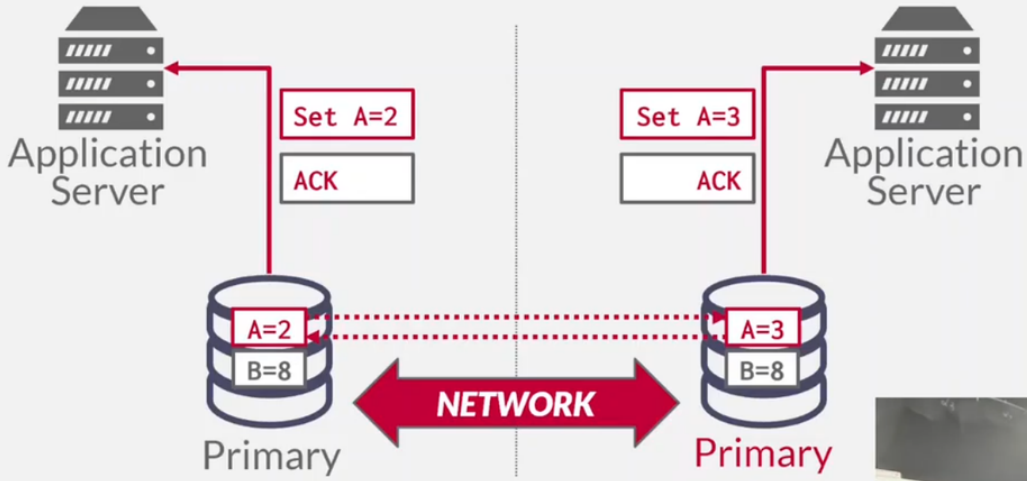
Partition Tolerance

- Split brain

两个节点通信断开，副本节点选举自己变成主节点并进行数据更改；然后两个节点恢复通信，两个节点均认为自己是主节点

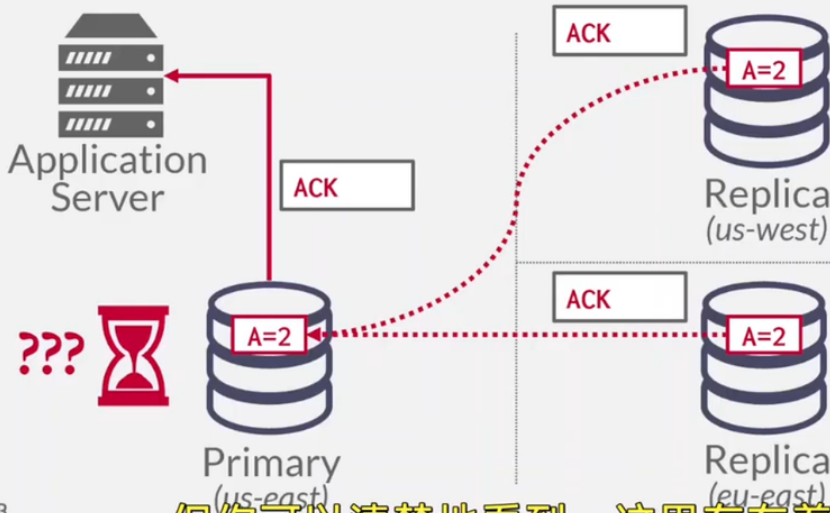
NOSQL解决方案：类似多版本控制，提出了向量时钟，直接选取版本号最高的值作为查询结果

PARTITION TOLERANCE



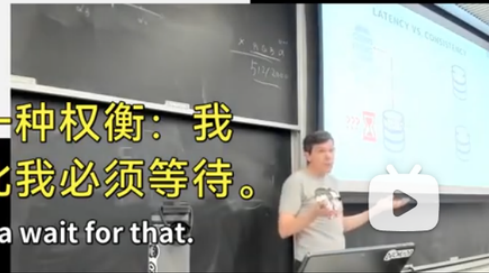
- 为了确保consistency，我们必须等待

LATENCY VS. CONSISTENCY



CMU-DB
IS-445/645 (Fall 2023)

但你可以清楚地看到，这里存在着一种权衡：我可以确保一切都很强健且一致，但为此我必须等待。
everything's strong and consistent, but I'm gonna wait for that.



传统数据库要求必须有主节点重连才能进行更新

NoSQL一般选择最新的更新

CAP/PACELC FOR OLTP DBMSs

How a DBMS handles failures determines which elements of the CAP theorem they support.

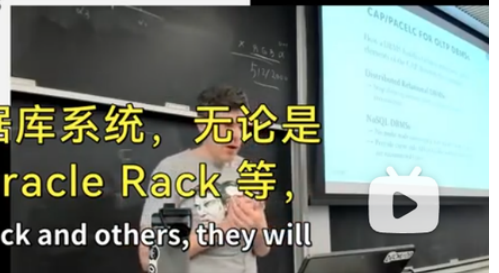
Distributed Relational DBMSs

→ Stop allowing updates until a majority of nodes are reconnected.

NoSQL DBMSs

- No multi-node consistency. Last update wins (*common*).
- Provide client-side API to resolve conflicts after nodes are reconnected (*rare*).

正如我之前所说，大多数分布式关系数据库系统，无论是传统的，如 20 世纪 80 年代的 DB2、Oracle Rack 等，ones, like the ones from the 1980s, like DB2 and Oracle Rack and others, they will



GOOGLE SPANNER

Google's geo-replicated DBMS (>2011)

Schematized, semi-relational data model.

Decentralized shared-disk architecture.

Log-structured on-disk storage.

Concurrency Control:

→ Strict 2PL + MVCC + Multi-Paxos + 2PC

→ **Externally consistent** global write-transactions with synchronous replication.

→ Lock-free read-only transactions.

他们构建了 Spanner 来运行他们庞大的广告基础设施。

They built Spanner for running their behemoth ad infrastructure.



通过全局唯一的时间戳来确保事务的顺序，时间戳由每个数据中心的原子钟和GPS接收器组合生成

SPANNER: CONCURRENCY CONTROL

MVCC + Strict 2PL with Wound-Wait Deadlock Prevention

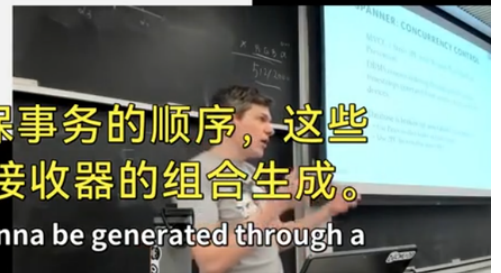
DBMS ensures ordering through globally unique timestamps generated from atomic clocks and GPS devices.

Database is broken up into tablets (partitions):

- Use Paxos to elect leader in tablet group.
- Use 2PC for txns that span tablets.

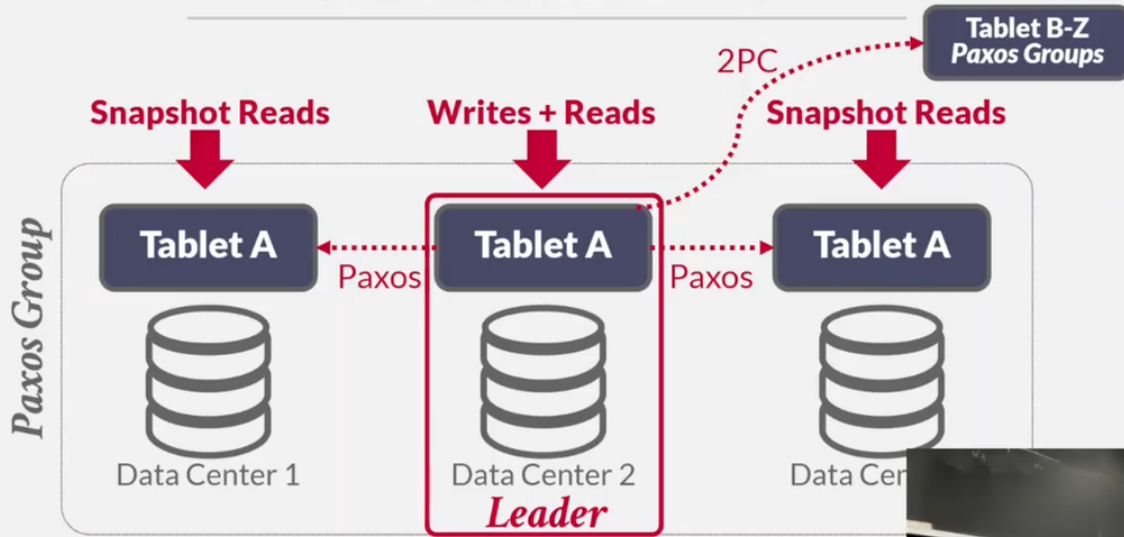
其工作原理是通过全局唯一的时间戳来确保事务的顺序，这些时间戳将由每个数据中心的原子钟和 GPS 接收器的组合生成。

transactions through globally unique timestamps that are gonna be generated through a



一个事务需要更新另一个tablet组，需要使用两阶段提交来执行更新操作；这些更新随后被传播到其他tablet，进而传递给它们的leader；leader通过paxos算法来保证consistency

SPANNER TABLETS



CMU-DB
IS-445/645 (Fall 2023)

这是一个很好的例子，说明事物并非总是互斥的，比如 Paxos 算法和两阶段提交协议。

So this is a good example where the things aren't mutually exclusive, like Paxos and



如何确保严格串行化和时间采样

SPANNER: TRANSACTION ORDERING

DBMS orders transactions based on physical "wall-clock" time.

- This is necessary to guarantee strict serializability.
- If T_1 finishes before T_2 , then T_2 should see the result of T_1 .

Each Paxos group decides in what order transactions should be committed according to the timestamps.

- If T_1 commits at $time_1$ and T_2 starts at $time_2 > time_1$, then T_1 's timestamp should be less than T_2 's.

这基本上说明了如何确保严格串行化以及进行时间采样。
This basically says how to guarantee strict serializability and do time sampling.



