本文提出了 PALF（Paxos-backed Append-only Log File）一种由 Paxos 支持的Append-only日志文件系统。PALF 的基本思想是将日志系统与整个数据库共同设计，以支持特定于数据库的功能，并将这些功能抽象为 PALF 基元，以支持其他分布式系统。

为解决上述问题，我们的设计选择是将共识协议与WAL模型集成在一起。在 WAL 模型中，数据库使用本地文件系统接口写入日志，与 （复制状态机）RSM 模型相比，记录和应用操作的顺序可以颠倒。写操作直接应用于数据库的存储引擎，然后生成并刷新重做日志（操作）。因此，事务大小的上限扩大了，读取请求只需访问存储引擎即可。然而，要设计这样一个复制日志系统，并为支持 WAL 模型提供类似本地文件系统的保证，仍然面临着以下挑战：

Leader Election：在实际部署中，数据库Leader通常与日志系统Leader同处一地，以减少延迟。在选举日志系统的Leader时，应考虑数据库的要求，例如，应优先选举与应用系统位于同一区域/IDC 的副本为Leader。不过，一个副本能否被选为Leader传统上取决于共识协议本身。Raft 中提出了领导权转移扩展，但它依赖于外部协调者主动将领导权转移给指定的副本，这损害了分布式数据库的可用性。

Uncertain Replication Results.：在 WAL 模型中，事务是提交还是中止取决于其提交记录是否已被持久化。如果日志系统因异常（如Leader转移）而向事务模型返回错误结果，数据可能会变得不一致。本地文件系统确实会返回明确的写入结果。然而，大多数共识协议实现在出现异常时不会返回明确的复制结果。例如，由于临时网络错误，先前的Leader已转变为follower。如果先前的Leader在退休前没有收到某些传输中日志的确认，它就无法感知这些日志是否已被多数人提交。因此，事务处理可能会卡住，因为事务引擎无法确定其提交记录是否已被持久化。

Data Change Synchronization：日志就是数据库，物理日志同步是将数据变化从数据库导出到下游系统的最常用方法之一。例如，物理备用数据库通过向备用数据库传输和应用redo日志，提供主数据库的相同副本。与直接复制日志文件不同，分布式数据库中的日志复制在将日志从主数据库中的一个复制组同步到备用数据库中的下游组方面存在挑战，而且这些复制组应该是独立可用的。一些复制协议将特定于群集的信息（如成员资格）嵌入日志，这就破坏了数据变化的连续性，并使下游复制组无法独立地重新配置群集。

Performance：性能。对于许多日志复制系统来说，单个复制组的吞吐量是有限的。因此，它们采用多个复制组，通过并行写入来提高总体吞吐量 [13, 15, 31]。然而，多个复制组可能会产生额外的开销。数据库中的一个数据分区通常与一个复制组绑定 [9, 42, 46]；复制组越多，意味着数据分区越小。这将导致更多的分布式事务，并降低整个数据库的性能[41]。

PALF提供典型的append-only日志接口，因此，数据库与 PALF 的交互就像与本地文件的交互一样。

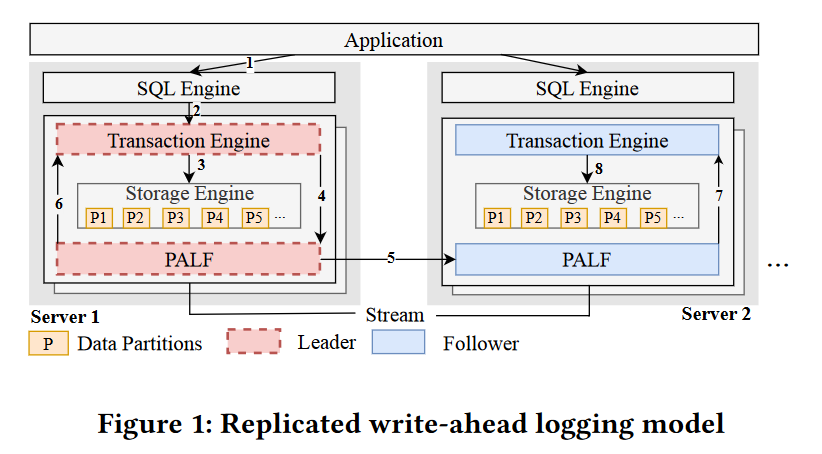
首先，PALF 将Leader选举与共识协议分离，以支持与数据库相关的选举优先级。例如，更接近上层应用程序的数据库副本可通过配置其选举优先级被选为Leader。由于独立选举的结果，PALF 引入了**日志重新确认（log reconfirmation stage）**阶段，以确保正确性。

其次，除非 PALF 的Leader崩溃，否则 PALF 会向日志写入器（数据库）返回明确的复制结果，这使得 PALF 就像一个本地文件。日志写入器（数据库）将收到 PALF 是否已提交日志的通知，即使之前的Leader已失去领导权。为此，在共识协议中引入了一个新的角色转换阶段--待定fallower，以确定待定日志的状态。在收到新Leader的日志之前，前Leader的角色不会切换为fallower。在此之后，事务的状态可以向前推进。例如，如果事务的提交记录没有被新Leader持久化，前任Leader就会回滚该事务。

此外，为了同步分布式数据库之间的数据变化，下游 Paxos 组被抽象为主 Paxos 组的镜像。它只接受主组的日志，并可独立重新配置。在 OceanBase 中，该功能用于将主数据库的重做日志同步到备用数据库。

最后，为了减少分布式事务带来的开销，我们将日志复制组的数量限制在集群中服务器的数量。复制组越少，单个复制组的吞吐量就越高，因为它要处理多个分区的日志。我们通过管道复制、自适应组复制和无锁写入路径pipeline replication, adaptive group replication, and lock-free write path.等系统优化措施，最大限度地提高了写入性能。

复制WAL模型



如图1所示，事务直接修改内存存储引擎中的数据（步骤2-3）。因此，事务规模的上限大大扩展，仅受存储引擎容量的限制。生成日志并追加到palf（4）Leader的事务引擎将PALF视为本地日志文件系统，它只关心日志记录是否已被刷新。 PALF 的职责是将Leader中执行的修改复制到fallower（步骤 5）。如果日志已被 PALF 提交，Leader将通知事务引擎结果（步骤 6），fallower将重放Leader对其自身执行的修改（步骤 7-8）。

PALF架构

如图 2 所示，PALF 是一个复制日志系统，由多个称为 PALF 组的复制组组成。在每个 PALF 组中，多个 PALF 副本放置在不同的服务器上以实现容错。事务引擎可以将日志追加到 PALF 组并从中读取日志，就像普通的追加文件一样。 PALF 由三个主要层组成：

interface layer（接口层）

PALF replicas layer（PALF 复制层）

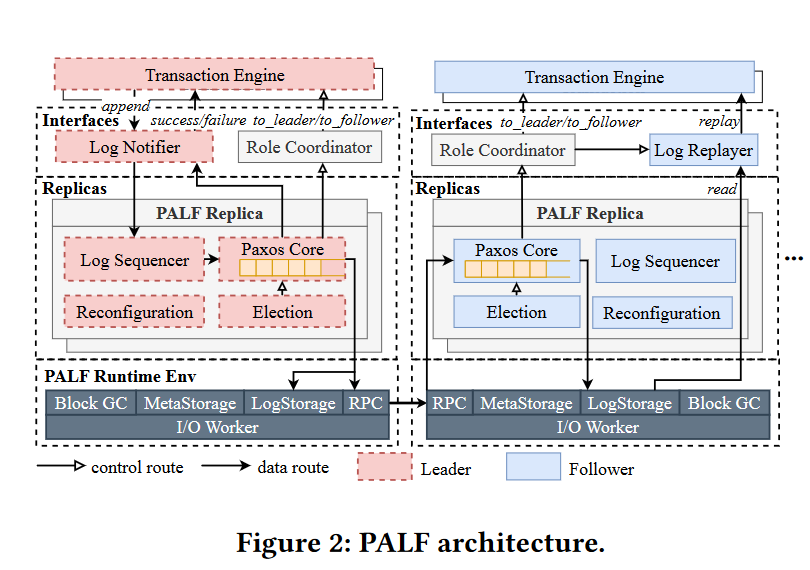
PALF runtime environment（PALF 运行环境）

下两层负责日志复制、重新配置和日志存储；上一层提供用户界面，并协调 PALF 和事务引擎的状态。

对于每个 PALF 组，事务引擎生成的记录会首先追加到leader中。日志排序器log sequencer将为每条记录分配一个单调递增的日志序列号 (LSN)，该序列号唯一标识 PALF 组内的日志条目。记录将封装为日志条目，并通过 Paxos 协议按照 LSN 的顺序复制到其他 PALF 复制器（fallower）并由其持久化。只有当大多数 PALF 复制已持久化日志条目时，才会提交该日志条目。并且PALF的选举是独立的。重新配置模块Reconfiguration管理 PALF 小组的成员资格。

在每台服务器上，都会激活一个名为 PALFEnv 的 PALF 运行环境，为 PALF 复制提供远程过程调用（RPC）接口并管理磁盘资源。具体来说，PALF 复制中的所有日志条目都以多个恒定大小的块的形式存储在 LogStorage 的一个唯一目录中。MetaStorage 存储元信息，如所有 PALF 复制的成员信息。BlockGC 负责在不再需要日志块时对其进行裁剪。PALF 复制发出的所有 I/O 请求都由 PALFEnv 中的统一 I/O Worker 池处理。

我们将 PALF 与事务引擎之间的交互概括为接口层。这就隔离了数据库特性对 PALF 的影响，提高了 PALF 的通用性。Leader中的日志通知器log notifier会通知事务引擎日志是否提交。follower中的日志重放器log replayer向事务引擎重放日志条目中记录的变化。如果 PALF 复制的role发生了转换（即Leader转换为fallower或fallower转换为Leader），它将向role coordinator抛出一个角色转换信号，role coordinator转发信号以转换事务引擎的角色。



系统接口

文本, 信件

描述已自动生成

图 3 显示了一组与数据相关的应用程序接口，省略了用于系统管理（如引导和重新配置）的接口。PALF 提供两种写日志的方法：追加和镜像append and mirror。追加方法将记录 r 提交给 PALF 组的Leader，它返回一个 LSN 以标识日志条目。日志条目的 LSN 是单调递增的，它代表日志块上日志条目的物理偏移量。变更序列号（CSN）是另一个日志条目标识符，将在第 4.2 节中讨论。为了实现高吞吐量，追加方法是异步的。当它返回时，事务引擎只需保证日志条目已被分配了唯一的 LSN 并提交给Leader的缓冲区。事务引擎将通过回调函数 AppendCb 获悉日志条目是否已提交。具体来说，当日志条目被提交时，成功方法将被调用，这意味着它已被大多数副本持久化，并且不会丢失；否则，失败方法将被调用。PALF 保证日志条目的回调函数最多被调用一次。镜像方法提供了另一种向 PALF 写日志的方法，它是为 PALF 组的镜像而设计的，只接受另一个 PALF 组提交的日志条目。这两种方法中，只有一种能同时向给定的 PALF 组写入日志（见第 5.2 节）。

读取方法可通过给定的 LSN 随机访问日志条目。提供了一种定位方法，可将更改序列号csn映射到日志序列号lsn。为便于在所有副本（如日志重放器）中实时消耗日志，提供了 monitor\_tail 方法来注册一个回调函数 TailCb，以监控 PALF 组的尾部。当提交新日志时，PALF 复制会调用 tail 方法，通知日志消费者当前日志的尾部。RoleCb 函数用于协调 PALF 复制和事务引擎的角色。当 PALF 复制的角色从Leader切换到fallower（从fallower到Leader）时，将调用 to\_follower (to\_leader) 方法。最后，trim 方法旨在指出给定 LSN 之前的无用日志条目。BlockGC 负责回收这些日志。

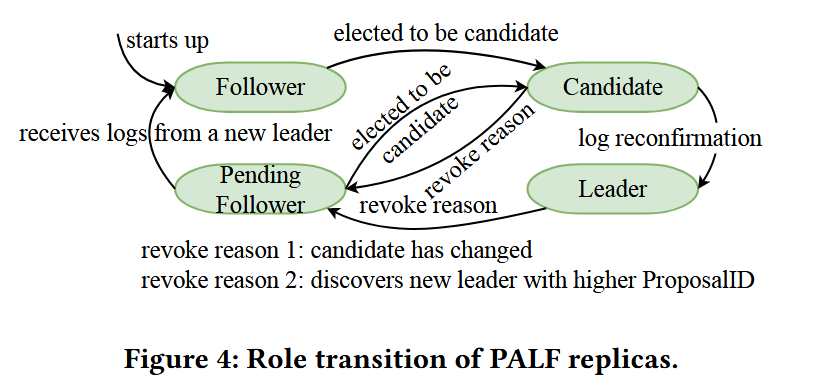
共识实现

PALF 采用强领导方法实现 Paxos，为了简单起见，它保留了 Raft 的日志复制功能。与 Raft 不同的是，PALF 将Leader选举与共识协议解耦，从而在不牺牲可用性的情况下操纵数据库Leader的位置。更多差异将在第 7 节中总结。

**领导选举的要求**。Raft 提供了领导力转移扩展功能，以操纵Leader的位置。不过，只有在前任领导人和期望领导人都健在的情况下，领导力转移扩展才会起作用。如果前一个Leader崩溃了，副本能否当选Leader完全受限于它存储的日志，而不是用户的愿望。如果一个不受欢迎的副本被选为新的Leader，这就需要外部coordinator来检测故障并执行领导权转移操作。如果coordinator崩溃，这种方法可能会给数据库带来可用性风险。

为了解决这个问题，PALF 将Leader选举与共识协议分离开来。用户可以灵活指定当选Leader的副本的优先级。如果前一个Leader崩溃，优先级第二高的副本将优先被选为新的Leader，而无需任何外部操作。如果前一个Leader从故障中恢复，且其优先级仍高于当前Leader的优先级，则领导权可自动转回已恢复的副本。（总结：之前的需要两个leader（之间的后来的）都在线，当之前的leader不在线时，需要协调者不崩溃，且领导者的位置随机了。现在只要根据优先级就可以将leader进行转换）

**角色转变**：在任何时候，每个副本都处于四种角色之一：Leader、fallower、candidate, pending follower.。图 4 显示了这些角色以及它们之间的转换。复制启动时，其角色是follower。每个副本都会定期向选举算法询问candidate是否是自己。如果fallower发现自己已成为候选者，它就会切换到candidate角色，并执行log reconfirmation，然后以普通Leader的身份接管工作。我们将在下文中介绍为什么需要log reconfirmation及其流程。如果Leader发现候选者不再是自己，或者Leader收到了来自新Leader的信息，它就会取消自己的领导权，转为pending follower。切换到pending follower而不是fallower的原因是，在领导权转移之前，事务引擎可能已经向Leader追加了一些日志。要确定这些待处理日志的复制结果，前一个Leader必须进入pending follower.角色，等待新Leader的日志（§4.1）。在所有待处理日志的状态明确后，将切换为fallower。



领导选举：候选者由基于租约的选举算法选出，该算法可确保不会有两个或两个以上的副本同时被选为候选者。用户可以为副本分配不同的选举优先级，选举算法保证大多数副本中优先级最高的副本将被选为候选者。选举算法本质上是基本 Paxos 算法的一种变体，它能在多数情况下就哪个副本拥有最高优先权达成共识。为确保这一点，每个副本在提出新一轮选举之前，都会尝试向所有副本广播自己的优先级；副本将响应在一定时间内收到的所有请求中优先级最高的请求。此外，选举算法还在每个副本中使用单调计时，以保证在一定时间内，如果有任何多数副本存活，候选人就会当选。

文本

描述已自动生成

**日志重新确认**。由于Leader选举灵活（自己确定优先级），候选者的日志数量可能少于其他副本。在接任Leader之前，候选者应重新确认上一任Leader添加的日志，以保证其日志不少于多数中的任何副本。Log Reconfirmation.（Alg.1）本质上是基本 Paxos 的一个完整实例。具体来说，候选者会通过 Paxos 准备信息（第 2 行）向所有副本广播高级提案 ID（Leader任期的标识符）pid + 1。每个副本都将存储 "准备 "报文的 pid，只有当 "准备 "报文中的 pid 大于副本所见过的最大 ProposalID pidmax 时，副本才会用日志回应候选者（第 9 行）。为避免传输无用的日志，准备信息的回执只包含日志的尾部 LSN。

一旦候选者从任何多数副本获得投票（第 3 行），它就会开始 Paxos 接受阶段：选择日志最长的副本（第 4 行），从该副本获取日志（第 5 行），并将这些日志复制到所有副本（第 6 行）。最后，候选者会向所有副本复制一份 StartWorking 日志（第 8 行）。请注意，StartWorking 日志是一种特殊的重新配置日志（§5.3），用于回滚上一任领导者可能未承诺的成员资格。只要 StartWorking 日志达到多数，候选者就会成为领导者。

**日志复制**。一旦Leader成功接管，它就会承担起复制和提交日志的责任。简而言之，日志条目按照 LSN 的顺序依次被添加到Leader、被Leader复制、被fallower确认以及被Leader提交。当记录添加到leader时，log sequence将为每个日志条目分配一个日志序列号 (LSN)。LSN 表示日志条目存储在日志块中的物理偏移量。如图 5 所示，LogStorage 中的日志块连续存储日志条目，下一个日志条目的 LSN 等于当前日志的 LSN 与日志大小之和。这种用 LSN 识别日志条目的方式使客户可以像操作普通文件一样操作 PALF，并便于数据库中的重做日志使用。当日志条目到达fallower时，fallower不会接受它，直到前面的日志都被接受为止。如果现有日志与提案号更高的新日志发生冲突，PALF 会以与 Raft 相同的方式截断冲突日志。这种机制建立了日志匹配属性 [34]。

正确性：利用 Raft 中的安全论证，我们只需验证 PALF 是否确保了以下属性：election safety, leader append-only, log matching, leader completeness, and state machine safety.选举安全、Leader追加、日志匹配、Leader完整性和状态机安全。PALF 借鉴了 Raft 的日志复制方案，因此自然而然地提供了leader append-only, log matching。Raft 和 PALF 的主要区别在于日志重新确认。候选者会执行一个 Basic Paxos 实例，从接受日志最多的副本中学习缺失的日志。如果上一任Leader提交的日志条目已被多数人接受，那么候选者就必须看到并学习该日志条目，从而确保选举的安全性和Leader的完整性election safety and leader completeness。我们将statemachine safety状态机安全属性的证明留在第 4.1 节，因为它与事务引擎应用日志的方式有关。

与交易引擎的互动

本节将介绍 PALF 在实现共识协议的基础上为 OceanBase 数据库事务引擎设计的功能。

显式复制结果

如果由于网络故障导致领导权转移，前任Leader可能不清楚追加日志条目是否已提交，这些日志被称为pending logs。pending logs可能会给事务引擎带来复杂性。 例如，事务引擎会生成事务的提交记录，并将其追加到 PALF 中。如果Leader意外失去领导权，事务引擎必须根据提交记录是否已被持久化来决定是提交还是回滚事务。

PALF 保证，除非Leader崩溃或网络永久中断，否则事务引擎将明确收到复制结果的通知。Leader负责提交日志和通知结果。如果Leader已被转移到另一个副本，先前的Leader会将自己切换为pending follower。当pending follower从新的Leader处收到日志时，待定日志的复制结果就会变得明确（提交或截断）。已提交日志的复制结果将通过调用 success 函数通知，截断日志将通过调用 failure 函数通知复制失败。 这就是为什么先前的Leader必须切换为待定fallower，并在成为fallower之前等待新Leader的日志。 对于每条记录，只会调用一个回调函数，且最多调用一次。

如图 6a 所示，前任Leader A 已将 log1、log2 和 log3 添加到副本 A，但尚未复制。随后，A 与 B 和 C 暂时分隔开来，C 被选为Leader，并从 B 处重新确认了 log2（图 6b）。 网络恢复后（图 6c），副本 A 已转变为待定fallower，因为其选举租期已过。它从新Leader C 处收到日志，A 中的 log2 将被接受，因为它已被新Leader提交。因此，副本 A 会调用 success 函数，将 log2 的复制结果通知数据库。相反，A 中的 log3 将被截断，因为 C 中的另一个 log3 包含了更大的 ProposalID。前任Leader A 复制的 log3 未能达成共识，因此副本 A 调用 failure 函数通知数据库。

请注意，如果图 6a 中的Leader A 崩溃，PALF 不需要直接通知数据库复制结果，因为内存中的所有状态都会丢失。当开始恢复时，事务引擎会切换到fallower，接收来自新Leader的日志，并复制本地日志以恢复事务。如果网络中断，先前的Leader就无法提供复制结果，因为新的Leader无法联系到它。如果网络恢复，前一个Leader就可以接收日志，并确定飞行中的日志是否已提交。

显式复制结果确保了共识协议的状态机安全属性。如果日志已提交，那么它一定已应用于Leader，并将重放给fallower。如果 PALF 未能就日志达成共识，上一个Leader的状态机将通过调用失败函数回滚。

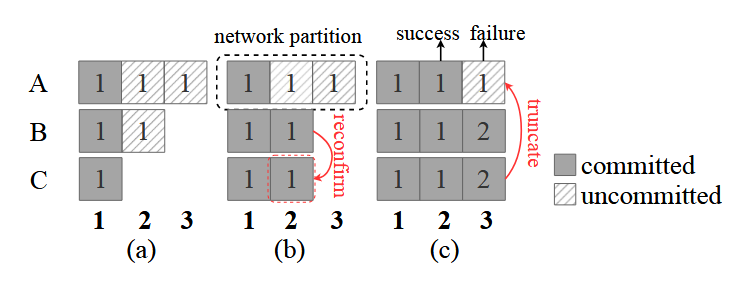


图 6：上一Leader A 向数据库返回显式复制结果。每个方框中的数字是添加日志时Leader的提案 ID。

更改序列号

数据同步工具（如变更数据捕获）通常按日志顺序消耗事务；但是，LSN 无法跟踪事务的顺序，因为事务是在单个 PALF 组内本地排序的。出于可扩展性考虑，数据分区通常分布在多个数据流中。如果不同的事务修改了不同流中的数据分区，其日志的 LSN 就无法比较。要跟踪 PALF 组中带有日志的事务顺序，一种自然的方法是在日志条目的有效载荷中记录事务的提交版本，有些系统就是这么做的 [18，42]。这种方法确实可行，但也有缺点。

例如，由于事务的并行执行（即为提交版本较高的事务日志分配较小的 LSN），提交版本可能不会严格随 LSN 增加。这种方法需要日志消费者解析日志的有效载荷，从而产生额外的开销。

PALF 提供另一种日志条目标识符，即更改序列号 (CSN)，它与 LSN 保持一致，并反映多个 PALF 组中的事务顺序。CSN 是一个 64 位整数，存储在每个日志条目的header中。PALF 不会推断 CSN 的含义，这意味着维护和识别 CSN 的开销非常小。当调用append方法时，Leader中的日志排序器会为日志分配 LSN 和 CSN。 PALF 维护 CSN 的不变性：在 PALF 组内单调递增。在每个 PALF 组内，对于任意两个日志 α 和 β，如果 α 的 LSN 大于或等于 β 的 LSN，那么 α 的 CSN 必须大于或等于 β 的 CSN。Formally,

LSNα ≥ LSNβ → CSNα ≥ CSNβ, ∀α, β.

该不变式保证了 CSN 在 PALF 组内单调增长，从而与 LSN 的顺序保持一致。 图 5 显示了日志的 LSN 和 CSN 之间的可视化关系。CSN 与日志条目一起保存。因此，即使Leader崩溃，上述不变式也始终有效。

PALF 提供了另一个关键保证：对于append方法，其输入参数 Ref CSN 和输出参数 CSN 始终遵循 CSN >= RefCSN。RefCSN 作为参考；该保证表示日志条目必须在 Re f CSN 发生事件后添加。因此，如果 Re f CSN 是一个具有全局意义的值，日志的 CSN 就能反映整个系统的顺序。

OceanBase 数据库使用 CSN 为事务提供全局意义的提交版本。当一个事务要提交时，事务引擎会从全局时间戳oracle中获取一个时间戳，并将带有时间戳的提交记录追加为Re f CSN。append方法返回的 CSN 会跟踪 Re f CSN 所指示的顺序，并作为事务的提交版本。请注意，CSN 不是由全局时间戳oracle生成的，全局时间戳oracle的值可能大于当前的全局时间戳。因此，未来从全局时间戳oracle中获取较小可读版本的读取请求可能无法看到该事务。为避免这种情况，事务引擎在全局时间戳大于 CSN 时才会响应客户端。通过事务引擎和 PALF 之间的合作，CSN 成功地跟踪了 PALF 组之间的事务顺序。

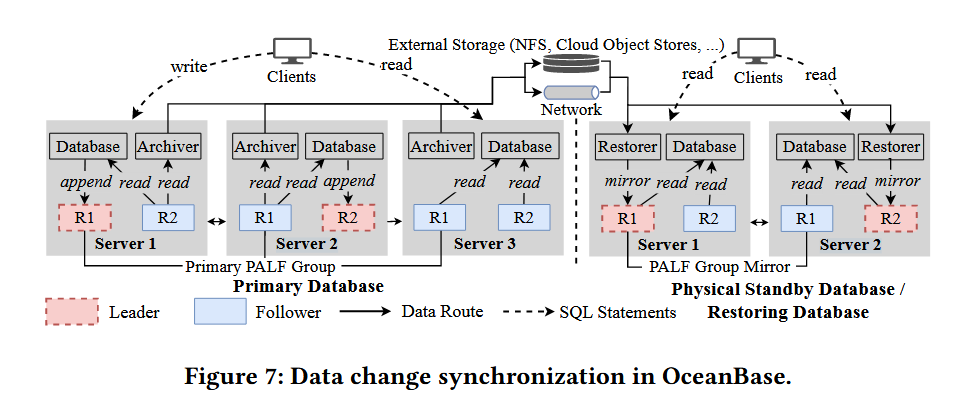
另一项受益于 CSN 的数据库功能是fallower读取。fallower读使fallower副本能够为只读请求提供最终一致性保证，从而减少延迟。T 的读取请求若能读取fallower中的完整数据，则要求 CSN 小于 T 的所有日志都已在fallower中重放，且任何未来写入的提交版本都应大于 T。CSN 的单调递增特性很自然地提供了这种保证。日志按照 LSN 的顺序复制和重放给fallower，这与 CSN 的顺序一致。 如果 CSN 为 T 的日志已在fallower日志中重放，则fallower日志的 CSN 必须大于 T。

5数据更改同步

除了为事务提供服务，分布式数据库还充当数据流的源头。下游应用程序可通过同步物理日志中记录的数据变化，提供各种服务。本节将介绍 OceanBase 中两个典型的物理日志同步场景，描述它们给 PALF 带来了哪些挑战，以及如何利用 PALF 的功能来应对这些挑战。

概况

当客户向数据库写入数据时，修改记录会追加到 PALF 组的leader，并复制到fallower。除了在数据库内复制日志外，还可将数据更改同步到数据库外，以实现更丰富的功能。OceanBase 数据库有两种典型的应用场景：物理备用数据库和数据库还原。



如图 7 所示，物理备用数据库是一个独立的数据库，其中的数据与主数据库完全相同。它可以满足部分读取请求，减轻主数据库的压力。与传统的主备架构相比，它具有更高的可用性，因为每个数据库集群都能承受故障。物理备用数据库最重要的特点之一是数据库级数据保护和灾难恢复，如果原主数据库不可用，物理备用数据库可以通过故障转移操作切换为主数据库。在生产数据库中，数据库还原是高可靠性功能的核心组成部分。 如果数据因存储介质损坏或人为错误而丢失，可使用离线存储（如 NFS 或云对象存储）中存储的归档日志来恢复相同的数据库。

物理备用数据库或数据库还原的基本思想类似：将主数据库或外部存储中的物理日志记录的数据变化同步到备用数据库或还原数据库。对于OceanBase数据库来说，实现这些功能的挑战之一是将日志从一个PALF组（或外部存储）同步到另一个PALF组。此外，这些 PALF 组应该是独立可用的。一个简单的解决方案是从主数据库的 PALF 组中读取日志条目，并将日志作为记录追加到备用数据库的相应 PALF 组中（图 3）。然而，共识协议会在追加记录上追加日志头以进行复制，这就导致主数据库和备用数据库的日志格式不一致。

PALF 组镜像

我们将跨 PALF 组同步数据变化的要求抽象为一个基本要素：PALF 组镜像，它是一个独立的 PALF 组，执行与第 3.4 节所述相同的共识协议。它维护着数据变更前缀的镜像，数据变更前缀存储在主 PALF 组或外部存储中。PALF 组镜像可独立重新配置，并根据需要切换为主组。

主 PALF 组和 PALF 组镜像之间最重要的区别之一是写日志记录的模式。在主 PALF 组中，日志记录被追加到 PALF 组，并附有日志头，然后通过共识协议复制到副本。至于 PALF 组镜像，它只接受主 PALF 组提交的日志。当提交日志被镜像到Leader时，日志头的某些字段（如 ProposalID）将被替换为Leader自己的值。Leader会重复使用原始日志条目的 LSN 和 CSN，将日志存储到 LSN，并复制日志给fallower。

为了区分主 PALF 组和镜像组，我们提出了两种访问模式 "主 "和 "镜像"。PALF 组的访问模式可通过故障转移或切换操作进行切换。问题是如何以原子方式向所有副本广播新的访问模式。显然，访问模式的原子广播等同于共识问题 [12]。因此，我们采用了基本的 Paxos 来切换 PALF 组的访问模式，并将每个副本的访问模式存储到 MetaStorage 中。

有了 PALF 组镜像基元，构建 OceanBase 数据库的数据变更同步架构就变得很自然了。如图 7 所示，物理备用数据库中的所有 PALF 组都是主数据库中 PALF 组的镜像，恢复数据库中的所有 PALF 组都是存储在外部存储中的数据变化的镜像。主 PALF 组提交事务日志后，日志归档器log archivers会从每个 PALF 组读取日志，然后将其存储到外部存储中或传输到备用数据库。在物理备用数据库中，恢复器restorers从归档器archiver接收日志（从外部存储中获取日志），并将日志映射mirror到 PALF 组镜像的Leader。日志由Leader提交后，将重放给所有副本（包括Leader）中的事务引擎。因此，在主数据库中执行的事务将同步到物理备用数据库；数据库还原执行类似的程序。

值得注意的是，备用数据库中事务引擎与 PALF 组之间的交互与主数据库不同。在备用数据库中，事务引擎执行标准的 RSM 模型，所有副本只需从 PALF 副本读取日志记录，并将数据变化重放至数据分区。因此，尽管 PALF 复制的角色可能是Leader，但事务引擎的角色始终是fallower。

独立重新配置

许多共识协议的实现[2, 15]都将重新配置命令作为普通日志条目进行存储和复制；然而，这种嵌入式方法可能会损害 PALF 群镜像的可用性。首先，事务引擎必须过滤无用的重新配置命令，因为这些命令只涉及它们写入的数据变化。其次，PALF 组镜像的成员资格与主 PALF 组的成员资格不同，因此应该能够独立进行重新配置。不过，PALF 组镜像只能接受主组的日志。 因此，物理备用数据库无法通过将重新配置命令写入普通日志记录的方式进行重新配置。

PALF 的重新配置类似于 Raft [33] 中描述的单服务器方法，每次只能添加或删除一个副本。Leader复制一个重新配置日志，记录新的成员资格，并与新成员资格确认一起提交。每个副本在收到更新的重新配置日志后，都会更新自己的成员资格。为使 PALF 组镜像能独立重新配置，PALF 将重新配置日志存储在 MetaStorage 中，与 LogStorage 分开。

独立元存储并不像想象中那么简单，它可能会损害共识协议的安全性，下面的例子展示了独立元存储带来的一些棘手问题。如图 8a 所示，Leader A 已提交了所有 LSN 小于等于 5 的日志，其中 A、D 和 E 拥有最新日志，但 B 和 C 落后。Leader A 希望通过将新成员复制到新成员中的副本，将副本 D 和 E 从组中移除；但可能会出现已提交日志丢失的安全风险。具体来说，一些已提交日志（3、4、5）尚未被任何多数新成员（A、B、C）刷新。如果副本 A 在 D 和 E 被删除后崩溃，这些日志就会丢失。图 8b 展示了在连续重组群集时可能出现的分裂脑[11]情况。通过移除副本 E 并添加副本 F 来用副本 F 替换副本 E 后，副本 B 和 F 可能会投票给 A，但副本 C 和 E 会投票给 D。

PALF 引入了对重新配置和Leader选举的限制，以解决这些异常现象。在重新配置时，Leader会向fallower发布新配置和日志屏障。 障碍是Leader发布配置时的日志尾数。每个fallower都会拒绝接受配置，直到其日志刷新到屏障之前。因此，副本 B 或 C 在接受屏障前的所有日志（3、4 和 5）之前，不会接受新配置（图 8a）。在选举首领时，PALF 会维护配置版本，以显示成员的版本；重新配置操作会递增配置版本。配置版本作为选举的主要优先级，因此副本 D 不会被选为Leader，因为其配置版本低于副本 C（图 8b）。 本文认为，尽管独立的元存储会给共识协议带来额外的复杂性，但它是有利的，因为它可以让日志消费者看不到元信息，并使 PALF 组镜像能够独立地重新配置集群。

性能优化

如第 2.2 节所述，事务引擎会将多个数据分区的大量重做日志强加给单个 PALF 组，这可能会使 PALF 成为瓶颈。本节将介绍如何系统地设计 PALF，以最大限度地提高 PALF 的性能。

流水线复制。为了提高吞吐量，PALF 利用现代多核处理器并发处理和复制日志。多个日志的共识相关状态被缓存在内存滑动窗口中，以避免 CPU 缓存缺失。 因此，多个日志的复制阶段可以重叠。

自适应群组复制 共识协议会给数据库带来额外的开销，其中包括至少两条网络信息（日志复制和确认）和一些 CPU 周期。在一个实例中批量处理多个日志条目是稀释共识产生的开销的常用方法。批处理日志的核心在于如何确定适当的批处理大小。 定期批处理日志或在 I/O 工作者空闲时立即批处理日志（反馈）是两种常见的方法。不过，前者可能会在低并发情况下产生额外的延迟。 后者可能会损害吞吐量，因为在高并发情况下，大量的小请求可能会压垮 I/O 设备。

PALF 采用自适应组大小复制日志，以平衡延迟和吞吐量。Leader在组缓冲区内缓存已添加的日志。冻结操作会将缓存日志打包到组日志条目中，然后复制给fallower。组日志条目中日志条目的数量（组因子）取决于冻结操作的执行频率。自适应分组复制的关键思路是在低并发时定期冻结日志，在高并发时根据 I/O 工作者的反馈冻结日志，但客户端的并发程度很难直接测量。假设有 n 个客户端同时向 PALF 追加记录 r。如果所有客户端的日志都已添加，但尚未提交，则这些缓存日志应立即冻结到组日志 gl 中，而无需等待恒定的时间间隔。因此，并发程度（n）与组因子（gf，缓存日志的数量）相关，这很容易通过计算日志来衡量。这种关系可形式化为

文本

描述已自动生成

实验表明，并发程度与 PALF 占用的 CPU 内核数成正比，因此组系数阈值可由硬件资源决定，而不是手动调整批量大小。如果分组因子小于阈值，则表示并发度较低，如果 I/O 工作者处于空闲状态，PALF 将冻结日志；否则，PALF 将以固定间隔（默认为 1 ms）冻结日志。与现有的批处理算法[20]相比，自适应分组复制足够简单且可预测，适合生产部署。

无锁写入路径 高并发性非但不会提高吞吐量，反而会在发生严重争用时降低性能。因此，我们为 PALF 设计了无锁写入路径，以避免线程间的竞争。写入路径的主要组件是日志排序器和分组缓冲器。日志排序器按顺序为日志条目分配 LSN。我们采用了无锁日志排序器，以避免其成为瓶颈。 当线程将日志添加到 PALF 时，它会以原子方式将 LSN 尾部的值加载到临时变量中，更新临时值，并通过原子比较和交换操作将其存储到 LSN 尾部。如果比较和交换操作因并发追加而失败，那么线程会重新加载 LSN 尾部，并再次循环获取 LSN [21]。

获取 LSN 后，多个线程会同时将日志条目填入组缓冲区。LSN 不仅是日志条目在磁盘上的地址，也是日志条目在组缓冲区中的偏移量，这意味着一个日志条目的预留缓冲区永远不会与另一个日志条目重叠。因此，组缓冲区不是竞争资源，多个线程可以同时将日志条目填充到不同的偏移量，而不会产生任何锁开销。