UNICORN

无感化跨链生态平台

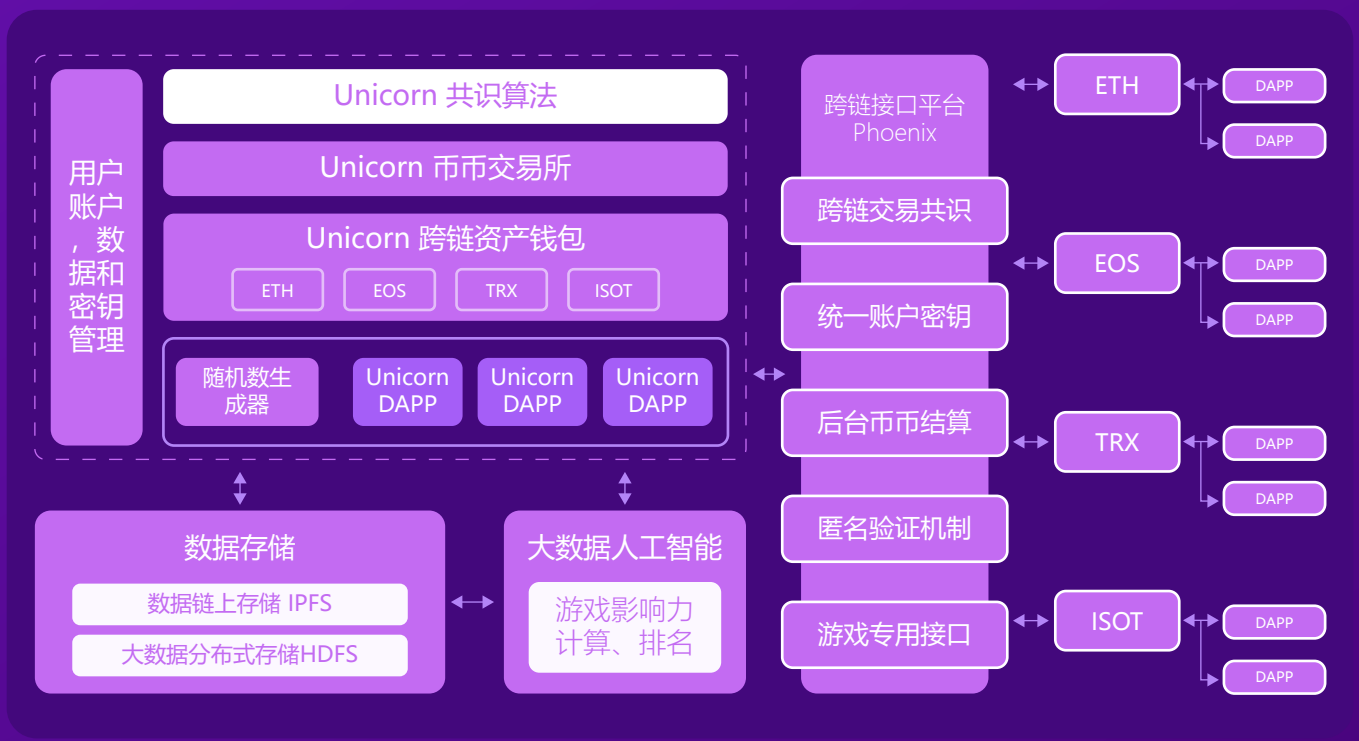
Non-probable cross-chain ecological platform

目录/CONTENTS

1. 简介

Unicorn专注于为平台用户提供无感化跨链服务，其中包括无感化跨链资产，无感化跨链应用和无感化跨链智能。具体来说，如下图所示，Unicorn主链由多个模块组成，主要包含用户账户管理模块，主链共识算法，后台交易所，多币种钱包，和主链DAPP。同时，Unicorn生态系统为主链模块提供数据和算力服务，其中包括，数存储服务，大数据计算和机器学习服务。

Unicorn生态系统同时提出跨链接口服务系统Phoenix，通过RPC远程调用协议，跨链接口可以调用主链服务。我们通过SDK和API的形式实现了，跨链共识，账户管理和后台币币兑换等服务，并且Phoenix平台有极好的可扩展性，为日后增加跨链平台提供方便接口。值得一提的是，Unicorn平台通过提供优化的智能合约库来降低开发DAPP的难度，使得本地和第三方DAPP生态能够快速扩张并且更快的完善。下面的章节我们将对各个模块和主要技术实现进行展开介绍。



2. Unicorn平台

以比特币和以太坊为代表的第一代区块链技术主要利用工作量证明（PoW）来防止服务被滥用，并在参与节点间达成共识。 PoW算法通常受到不对称特征的束缚，也就是说，工作必须有一定工作了才能解决，但容易验证工作量。广泛采用的基于PoW的区块链系统通常依赖各种散列函数，如SHA-256，Scrypt，CryptoNote，Ethash，Equihash等。然而，基于PoW的区块链协议，需要消耗大量的计算资源和电能。例如，计算出在比特币区块链上生成一个区块需要超过260次哈希操作，这将导致大量的能源消耗，有研究显示，比特币区块链的总能量消耗与一个小的国家相当。

因为除了保证区块链交易的安全性之外，这些散列函数在区块链系统中没有其他实际用途，因此学术研究人员提出了一种新的工作证明（PoS），相对于PoW，PoS可以显着提高能源使用效率，同时安全的实现分布式共识。相比于PoW中要求矿工投入计算资源以参与领导者选举过程，PoS改为运行一个程序，该程序随机地选择参与节点作为领导者，这个随机过程根据当前的区块链中每个节点所占有的股份分配不同的随机概率。

基于节点投票权，我们实现下面RAFT共识算法。

2.1 共识算法实现RAFT

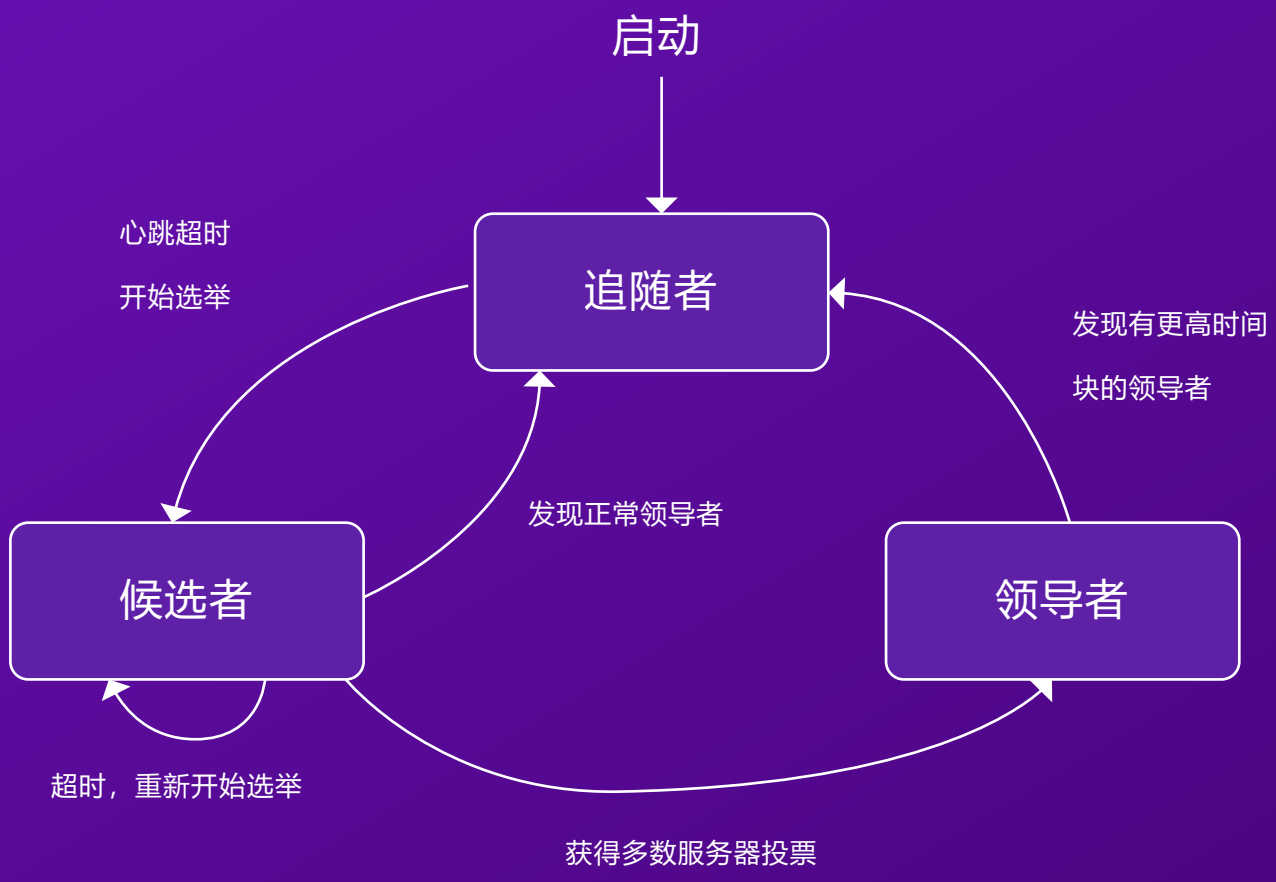
RAFT算法更强调保证各个节点内共享相同的日志数据状态，这无疑正是Unicorn所需要的；同时与其他共识算法相比，RAFT可以调整日志数据更新周期时间，这使得RAFT可以针对不同应用场景可以调整不同的TPS，而这也正是Unicorn应对不同游戏和应用场景所需要的。

2.1.1 Raft 基础

Raft集群包含多个服务器，由五台服务器组成的Raft是一个典型的配置，它可以容忍两个服务器发生故障。在任何给定时间，每个服务器处于以下三种状态之一：

- 领导者
- 追随者
- 候选者

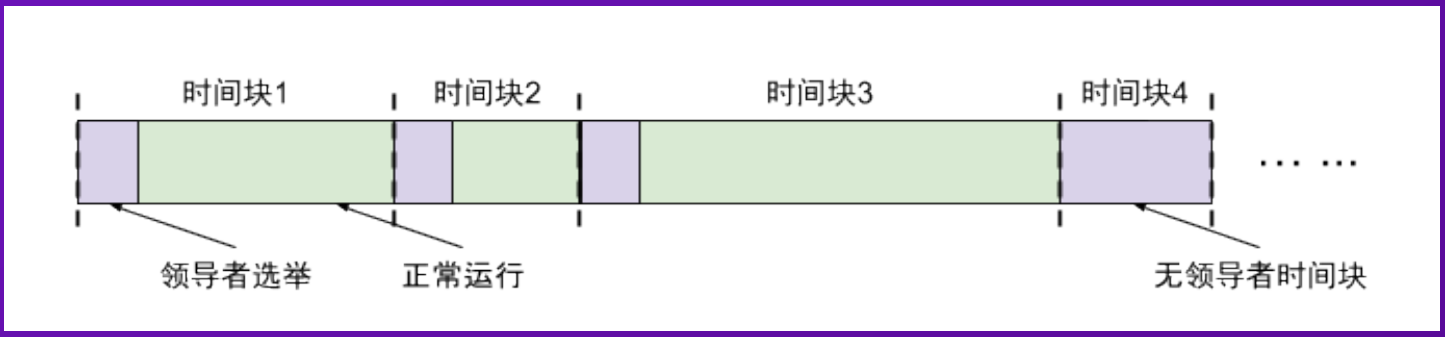
在正常运行中，只有一个领导者，所有其他服务器都是追随者。追随者是被动的：他们自己不发出请求，只是回应领导者和候选人的请求。领导者处理所有客户请求（如果客户请求了追随者，追随者将其请求重定向到领导者）。候选人状态用于选举新的领导者。下图显示了这些状态及其转换。



图：RAFT节点三种状态之间的转换

2.1 跨链钱包

Raft将时间划分为任意长度，如下图所示。时间块用连续的整数编号。每个时间块从选举开始，其中一个或多个候选人试图成为领导者。如果候选人赢得选举，那么它将成为其余任期的领导者。在某些情况下，选举将导致分裂投票。在这种情况下，该时间块将以没有领导者结束，不久将开始一个新的选举。Raft确保在给定的任何一个时间块内最多只有一个领导者。



图：Raft中时间块示例

不同的服务器可以在不同的时间观察时间块之间的转换，并且在某些情况下，服务器可能不会观察到选举甚至整个时间块。时间块在Raft中充当逻辑时钟[4]，它允许服务器检测过时的信息，例如陈旧的领导者。每个服务器都存储一个当前的时间块编号，该编号随着时间的推移单调增加。当服务器通信时，他们会交换当前时间块。如果一个服务器的当前时间块小于另一个服务器的当前时间块，则它将其当前时间块更新为两者中更大的那个值。如果候选人或领导者发现其时间块已过期，则会立即转化为追随者状态。如果服务器收到过期时间戳的请求，它将拒绝该请求。

Raft服务器使用远程过程调用（RPC）进行通信，并且一致性算法的实现仅需要两种类型的RPC。 RequestVote RPC由候选人在选举期间启动，AppendEntries RPC由领导者发起，以备份日志条目并提供简单的心跳容错机制。服务器如果没有及时收到响应，则重试RPC，并且它们并行发出RPC以获得最佳性能。

2.1.2 Raft 领导者选举

Raft使用心跳机制触发领导者选举。当服务器启动时，它们是追随者的身份。只要服务器从领导者或候选者接收到有效的RPC，该服务器就会保持追随者状态。领导者向所有追随者定期发送心跳（不带日志条目的AppendEntries RPC）以保持其领导者身份。如果追随者在一段时间内没有收到任何心跳信息，那么它就假定没有正在工作的领导者，然后开始选举新的领导者。

为了开始选举，追随者增加其当前时间块并转换到候选者状态。然后它投票支持自己并给集群中的每个其他服务器并行发出RequestVote RPC。候选人继续处于这种状态，直到发生以下三种情况中的一种：

1.它赢得选举

2.另一个服务器将自己确立为领导者

3.一段时间过去而没有获胜者

下面，我们对这三种情况进行详细讨论。

如果候选人在相同的时间块内收到来自集群中的大多数服务器的投票，则该候选人将赢得选举。每个服务器将按照先到先得的原则在给定时间块内投票最多一个候选人。大多数选票规则确保最多一个候选人可以赢得特定时间块的选举。一旦候选人赢得选举，它就会成为领导者。然后，它会向所有其他服务器发送心跳消息，以确定其领导者身份并阻止新的选举发生。

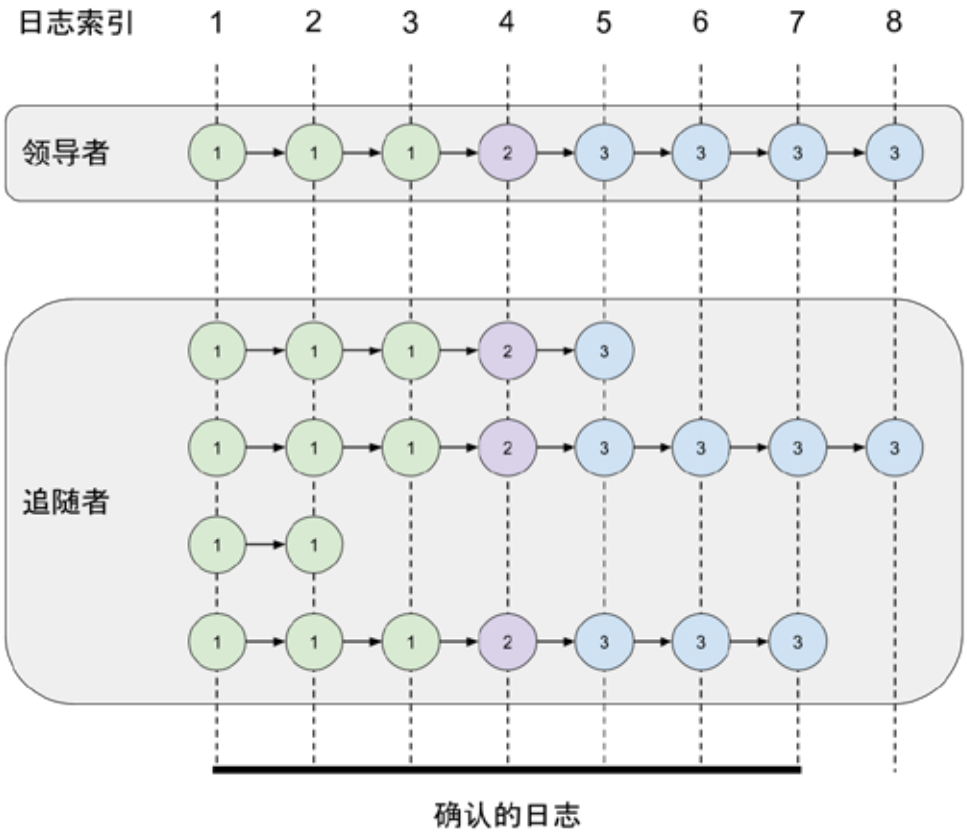
在等待投票时，候选人可能会从声称自己是领导者的另一台服务器收到一个AppendEntries RPC。如果领导者的时间块（包含在其RPC中）至少与候选人当前的时间块一样大，则候选人将该领导者视为合法并返回到追随者状态。如果RPC中的时间块小于候选人的当前时间块，则候选人拒绝RPC并继续保持候选状态。

最后一种可能的结果是候选人既没有赢得也没有输掉选举：如果多个追随者同时成为候选人，则余下追随者的投票将被分割，很大概率没有候选人获得超过半数的投票。当这种情况发生时，每个候选人将通过增加其时间块并启动新一轮的Request-Vote RPC来开始新的选举。但是，如果不采取额外措施，分割投票的情况可能会无限期重复。

Raft使用随机选举超时规则确保了分割投票出现的几率很小并且就算出现也可以快速得到解决。首先，为了防止投票分割，Raft从固定间隔（例如，150-300ms）的选举超时，扩展到随机的选举超时，这保证了在大多数情况下只有一台服务器会先超时，并且进入下一个时间块，因为其时间块比当前其他候选者的时间块都要大，那么它将在其他服务器超时之前赢得选举并发送心跳。

2.1.3 Raft 一致性

领导者被选出，它就开始为客户请求提供服务。每个客户端请求都包含由备份服务器执行的命令。领导者将命令作为新条目附加到其日志中，然后并行地将AppendEntries RPC与每个其他服务器进行交互以备份数据条目。当数据条目被安全地传输到大部分追随者时，领导者将条目应用到其状态机（数据库）并将该执行的结果返回给客户端。如果追随者宕机，运行缓慢，或者网络数据包丢失，领导者将无限地重试AppendEntries RPC（即使它已返回客户端），直到所有追随者最终存储所有日志条目。



图： 不同颜色的圆形代表了不同的时间块所同步的日志，日志本身也有一个自增的索引，我们可以看到该日志确认到了索引7，这是因为包含领导者在内，有大多数的服务器（3）确认了这个状态

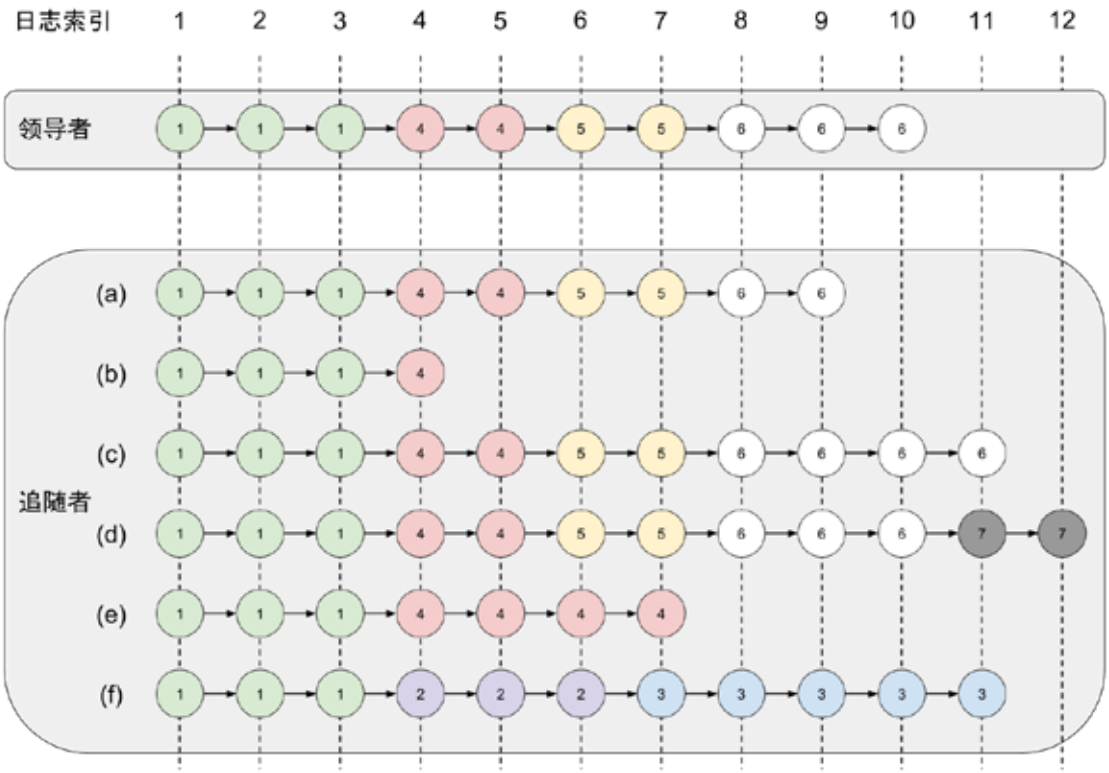
日志的组织如上图所示。每个日志条目都会在领导者收到条目时，存储状态机命令以及时间块编号。日志条目中的时间块数字用于检测日志之间的不一致。每个日志条目还有一个整数索引，用于标识其在日志中的位置。

领导者决定何时将日志条目安全的应用于状态机（数据库写入），这样的条目称为确认的日志。Raft保证提交的日志是容错的，并且最终将由所有可用的服务器执行。一旦创建日志条目的领导者将其复制到大多数服务器上（例如，上图中的日志索引7）那么这个日志即被确认。这也会确认领导者日志中的所有前面日志条目，包括由前任领导者创建的日志条目。领导者跟踪它知道要提交的最高索引，并在未来的AppendEntries RPC（包括心跳）中包含该索引，以便其他服务器匹配。一旦追随者得知确认的日志条目编号，它就会将该条目应用于其本地状态机（按日志顺序）。

我们设计了Raft日志机制，以在不同服务器上的日志之间保持高水平的一致性。这不仅简化了系统的行为并使其更具可预测性。 Raft维护以下属性：

- 如果不同日志中的两个条目具有相同的索引和时间块，则它们存储相同的命令。
- 如果不同日志中的两个条目具有相同的索引和时间块，则所有前面的条目中的日志都相同。

第一个属性遵循以下事实：领导者在给定的时间块中最多创建一个具有给定日志索引的条目，并且日志条目永远不会更改它们在日志中的位置。第二个属性由AppendEntries执行的简单一致性检查保证。发送AppendEntries RPC时，领导者在其日志中包含紧接在新条目之前的条目的索引和时间块。如果追随者没有在其日志中找到具有相同索引和时间块的条目，则它拒绝新条目。一致性检查充当归纳步骤：日志的初始空状态满足日志匹配属性，然后一致性检查在日志扩展时保证了日志匹配属性。因此，只要AppendEntries成功返回，领导者就会知道追随者的日志和自己所记录的最新日志相同。



当领导者被确定后，任何场景（a-f）都可能出现在追随者的日志中。每个圆形表示一个日志条目，圆形中的数字是它的时间块编号。 追随者可能缺少条目（a-b），也可能有额外的未提交条目（c-d）或两者都存在（e-f）。例如，（f）的产生是因为它是第2个时间块的领导者，但其在日志中添加了几个条目后，还没来得及提交和发送给其他追随者就宕机了，然后它很快重新启动，又成为第3个时间块的领导者，并在其日志中添加了更多.条目，在第2个或第3个时间块中的任何条目都在还没来得及提交和发送，服务器再次宕机并在下面几个时间块内保持宕机状态

在正常操作的情况下，领导者和追随者的日志将永远保持一致，因此AppendEntries一致性检查永远不会失败。但是，领导者宕机可能会使追随者日志的不一致（旧的领导者可能没有完全复制其日志中的所有条目）。这些不一致可能会导致一系列领导者和追随者宕机。上图说明了追随者日志可能与新领导者的日志不同的方式。追随者可能缺少领导者中存在的条目，可能具有领导者不存在的额外条目，或两者都有。日志中缺少和多出的条目可能跨越多个时间块。

在Raft中，领导者通过强制追随者把日志复制成自己的日志来处理不一致的情况。这意味着追随者日志中的冲突条目将被领导者日志中的条目覆盖。

要使追随者的日志与自己的日志保持一致，领导者必须找到两个追随者同意的最新日志条目，在该点被确认之后删除追随者日志中的所有条目，并向追随者发送所有领导者的条目。所有这些操作都是在响应AppendEntries RPC执行的一致性检查时发生的。领导者为每个追随者维护一个nextIndex，这是领导者将发送给该追随者的下一个日志条目的索引。当一个领导者刚被选举出来时，它会将所有nextIndex值初始化到其日志中最后一个之后的索引（上图中的日志索引11）。如果追随者的日志与领导者的日志不一致，则AppendEntries一致性检查将在下一个AppendEntries RPC中失败。在拒绝之后，领导者减少nextIndex并重试AppendEntries RPC。最终nextIndex将达到领导者和追随者日志匹配的点。发生这种情况时，AppendEntries将成功，这将删除追随者日志中的任何冲突条目，并附加领导者日志中的条目。一旦AppendEntries成功，追随者的日志将与领导者的日志一致，并且在剩下的时间块中，它将一直保持这种状态。

2.1.4 RAFT 状态机

在所有服务器上持久存在的：（在响应远程过程调用 RPC 之前稳定存储的）

名称	描述
currentTerm	服务器最后知道的任期号（从 0 开始递增）
votedFor	在当前任期内收到选票的候选人 id（如果没有就为 null）
log[]	日志条目；每个条目包含状态机的要执行命令和从领导人处收到时的任期号

在所有服务器上不稳定存在的：

名称	描述
commitIndex	已知的被提交的最大日志条目的索引值（从 0 开始递增）
lastApplied	被状态机执行的最大日志条目的索引值（从 0 开始递增）

在领导人服务器上不稳定存在的：（在选举之后初始化的）

名称	描述
nextIndex[]	对于每一个服务器，记录需要发给它的下一个日志条目的索引（初始化为领导人上一条日志的索引值 +1）
matchIndex[]	对于每一个服务器，记录已经复制到该服务器的日志的最高索引值（从 0 开始递增）
log[]	日志条目；每个条目包含状态机的要执行命令和从领导人处收到时的任期号

附加日志远程过程调用 （AppendEntries RPC）

由领导人来调用复制日志，也会用作 heartbeat

名称	描述
term	领导人的任期号
leaderId	领导人的 id，为了其他服务器能重定向到客户端
prevLogIndex	最新日志之前的日志的索引值
prevLogTerm	最新日志之前的日志的领导人任期号
entries[]	将要存储的日志条目（表示 heartbeat 时为空，有时会为了效率发送超过一条）
leaderCommit	领导人提交的日志条目索引值

返回值	描述
term	当前的任期号，用于领导人更新自己的任期号
success	如果其它服务器包含能够匹配上 prevLogIndex 和 prevLogTerm 的日志时为真

接受者需要实现：

1.如果 term < currentTerm返回 false

2.如果在prevLogIndex处的日志的任期号与prevLogTerm不匹配时，返回 false

3.如果一条已经存在的日志与新的冲突（index 相同但是任期号 term 不同），则删除已经存在的日志和它之后所有的日志

4.添加任何在已有的日志中不存在的条目

5.如果leaderCommit>commitIndex，将commitIndex设置为leaderCommit和最新日志条目索引号中较小的一个

投票请求 RPC（RequestVote RPC）

由候选人发起收集选票

名称	描述
term	领导人的任期号
candidateId	请求投票的候选人 id
lastLogIndex	候选人最新日志条目的索引值
lastLogTerm	候选人最新日志条目对应的任期号

返回值	描述
term	目前的任期号，用于候选人更新自己
voteGranted	如果候选人收到选票为 true

接受者需要实现：

1.如果term < currentTerm返回 false

2.如果votedFor为空或者与candidateId相同，并且候选人的日志和自己的日志一样新，则给该候选人投票

服务器需要遵守的规则：

所有服务器：

●如果commitIndex > lastApplied，lastApplied自增，将log[lastApplied]应用到状态机

●如果 RPC 的请求或者响应中包含一个 term T 大于 currentTerm，则currentTerm赋值为T，并切换状态为追随者（Follower）

追随者（followers）：

●响应来自候选人和领导人的 RPC

●如果在超过选取领导人时间之前没有收到来自当前领导人的AppendEntries RPC或者没有收到候选人的投票请求，则自己转换状态为候选人

候选人：

- 转变为选举人之后开始选举：
  - currentTerm自增
  - 给自己投票
  - 重置选举计时器
  - 向其他服务器发送RequestVote RPC
- 如果收到了来自大多数服务器的投票：成为领导人
- 如果收到了来自新领导人的AppendEntries RPC（heartbeat）：转换状态为追随者
- 如果选举超时：开始新一轮的选举

领导人：

- 一旦成为领导人：向其他所有服务器发送空的AppendEntries RPC（heartbeat）；在空闲时间重复发送以防止选举超时
- 如果收到来自客户端的请求：向本地日志增加条目，在该条目应用到状态机后响应客户端
- 对于一个追随者来说，如果上一次收到的日志索引大于将要收到的日志索引（nextIndex）：通过AppendEntries RPC将 nextIndex 之后的所有日志条目发送出去
  - 如果发送成功：将该追随者的 nextIndex和matchIndex更新
  - 如果由于日志不一致导致AppendEntries RPC失败：nextIndex递减并且重新发送
- 如果存在一个满足N > commitIndex和matchIndex[i] >= N并且log[N].term == currentTerm的 N，则将commitIndex赋值为 N

Raft 一致性算法的总结（不包括成员变化 membership changes 和日志压缩 log compaction）

性质	描述
选举安全原则（Election Safety）	一个任期（term）内最多允许有一个领导人被选上
领导人只增加原则（Leader Append-Only）	领导人永远不会覆盖或者删除自己的日志，它只会增加条目
日志匹配原则（Log Matching）	如果两个日志在相同的索引位置上的日志条目的任期号相同，那么我们就认为这个日志从头到这个索引位置之间的条目完全相同
领导人完全原则（Leader Completeness）	如果一个日志条目在一个给定任期内被提交，那么这个条目一定会出现在所有任期号更大的领导人中
状态机安全原则（State Machine Safety）	如果一个服务器已经将给定索引位置的日志条目应用到状态机中，则所有其他服务器不会在该索引位置应用不同的条目

Raft 算法保证这些特性任何时刻都成立

Raft 通过首先选出一个领导人来实现一致性，然后给予领导人完全管理复制日志（replicated log）的责任。领导人接收来自客户端的日志条目，并把它们复制到其他的服务器上，领带人还要告诉服务器们什么时候将日志条目应用到它们的状态机是安全的。通过选出领导人能够简化复制日志的管理工作。例如，领导人能够决定将新的日志条目放到哪，而并不需要和其他的服务器商议，数据流被简化成从领导人流向其他服务器。如果领导人宕机或者和其他服务器失去连接，就可以选取下一个领导人。

通过选出领导人，Raft 将一致性问题分解成为三个相对独立的子问题：

- 领导人选取（Leader election）： 在一个领导人宕机之后必须要选取一个新的领导人
- 日志复制（Log replication）： 领导人必须从客户端接收日志然后复制到集群中的其他服务器，并且强制要求其他服务器的日志保持和自己相同

●安全性（Safety）： Raft 的关键的安全特性是上表中提到的状态机安全原则（State Machine Safety）：如果一个服务器已经将给定索引位置的日志条目应用到状态机中，则所有其他服务器不会在该索引位置应用不同的条目。上文阐述了 Raft 是如何保证这条原则的，解决方案涉及到一个对于选举机制另外的限制，这一部分会在上文中说明。

在说明了一致性算法之后，本章会讨论有关可用性（availability）的问题和系统中时序（timing）的问题。

2.2 分叉解决机制 TPoS

由于网络分叉或者其他节点失效原因，RAFT共识出块可能产生分叉，对于出块分叉，我们提出以下TPoS树形块签名方案（Tree-like Proof of Stake）。简单来说，我们假设存在一组固定的验证人和一个建议机制（例如熟悉的工作提议机制证明），它产生现有块的子块，形成一个不断增长的块树。树的根部通常被称为“生成块”。在正常情况下，我们预计提案机制通常会在链接列表中一个接一个地提出块（即，每个“父”块具有恰好一个“子”块）。但是，在网络延迟或存在故意攻击的情况下，提案机制将不可避免地会产生同一父母的多个子女。TPoS的工作是从每个父母中选择一个孩子，从而从分块树中选择一个规范链。为了提高效率，TPoS只考虑构成检查点树的检查点的子树，而不是处理完整的块树（如下图所示）。生成块是一个检查点，并且块树（或块编号）中的高度为H的倍数的每个块也是检查点。块高度为H*k的块的“检查点高度”简单地称为k，检查点c的高度h(c)是检查点链中从父节点一直延伸到父节点的元素的数量（如下图所示）。

每个验证人首先都有一笔存款，兑换成Unicorn代币加入到区块链中，每位验证人的存款将随着奖励和处罚而升降。验证人可以广播如下表所示的包含四条信息的投票消息：两个检查点s和t连同他们的高度h(s)和h(t)。我们要求s是检查点树中的t的祖先，否则投票将被视为无效。如果验证人v的公钥不在验证人集中，则考虑投票无效。连同验证人的签名，我们将以<v,s,t,h(s),h(t)>的形式表示这些投票。

符号	说明
s	任何合理检查点的哈希值（“源”）
t	作为s的后代的任何检查点哈希（“目标”）
h(s)	检查点树中检查点s的高度
h(t)	检查点树中检查点t的高度
S	来自验证人v私钥的<v,s,t,h(s),h(t)>的签名

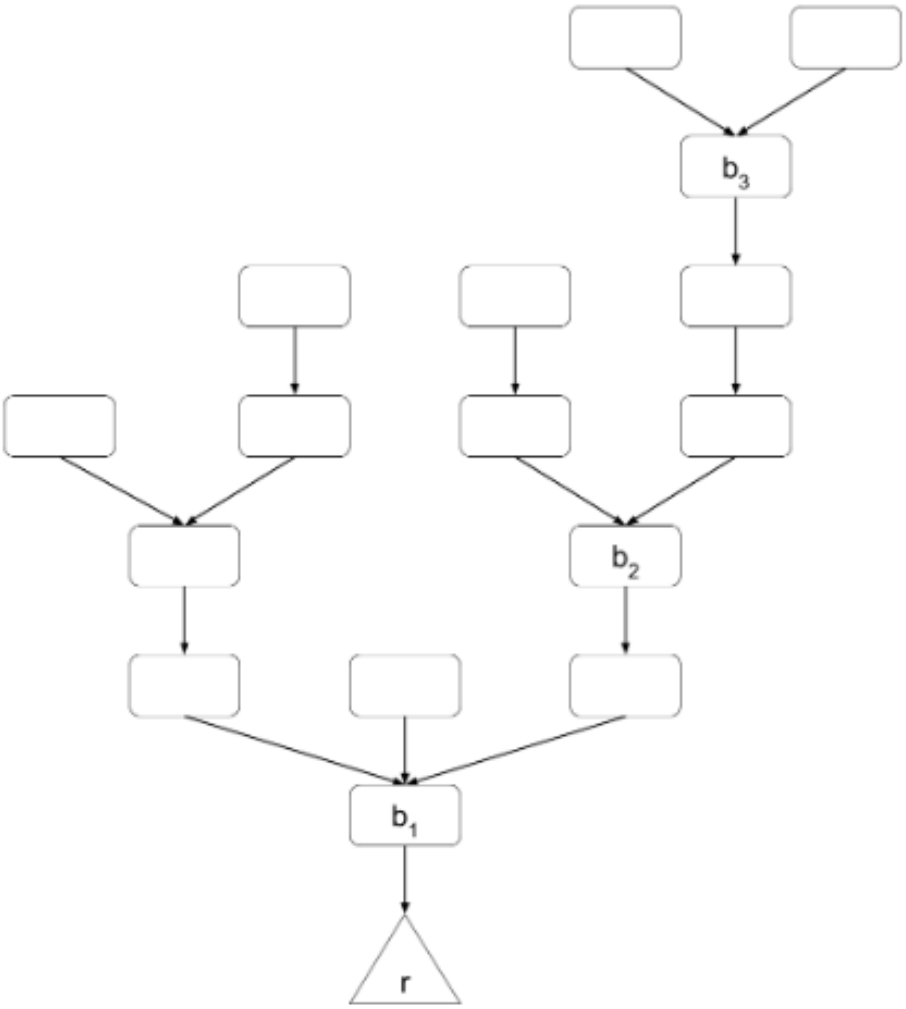
我们定义以下术语：

●绝对多数链接是一对有序的检查点（a，b），也写成a→b，这样至少有2/3的验证人（通过存款）已经发布了对源a和目标b的投票。绝对多数链接可以跳过检查点，即h(b)> h(a)+ 1。下图显示了黑色的三个不同的绝对多数链路：r→b1，b1→b2和b2→b3。

●当且仅当它们是不同分支中的节点时，两个检查点a和b才被认为冲突，即，a和b相互既不是祖先也不是后代。

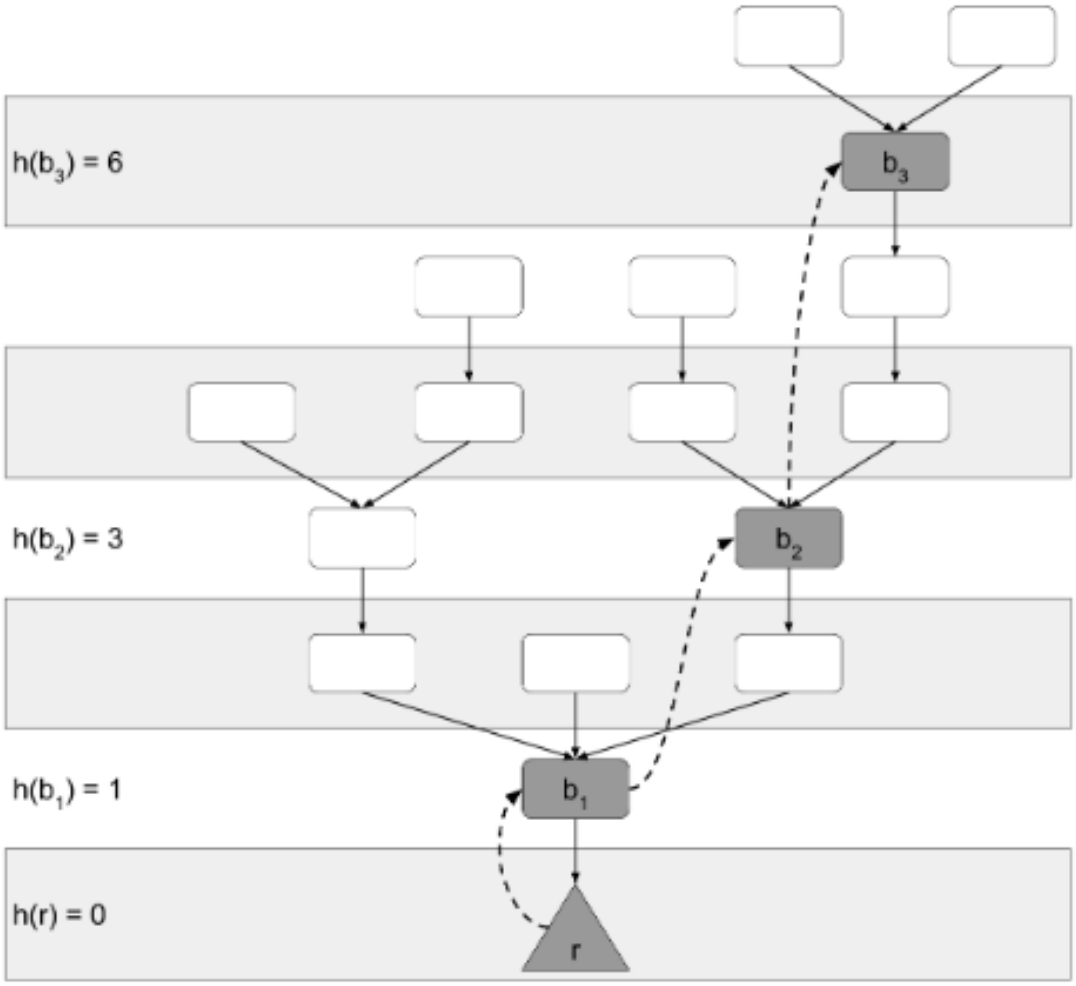
●如果这个节点是根节点或者存在绝对多数链接c'→c，其中c'是对齐的，则检查点c是对齐的。下图显示了四个对齐块的链。

●如果检查点c被证明是合理的并且存在绝对多数链接c→c'，其中c'是c的直接子项，则称其为最终确定。



图：块树说明图

检查点树，实线表示检查点之间的H个块，检查点由圆角矩形表示，树的根用“r”表示



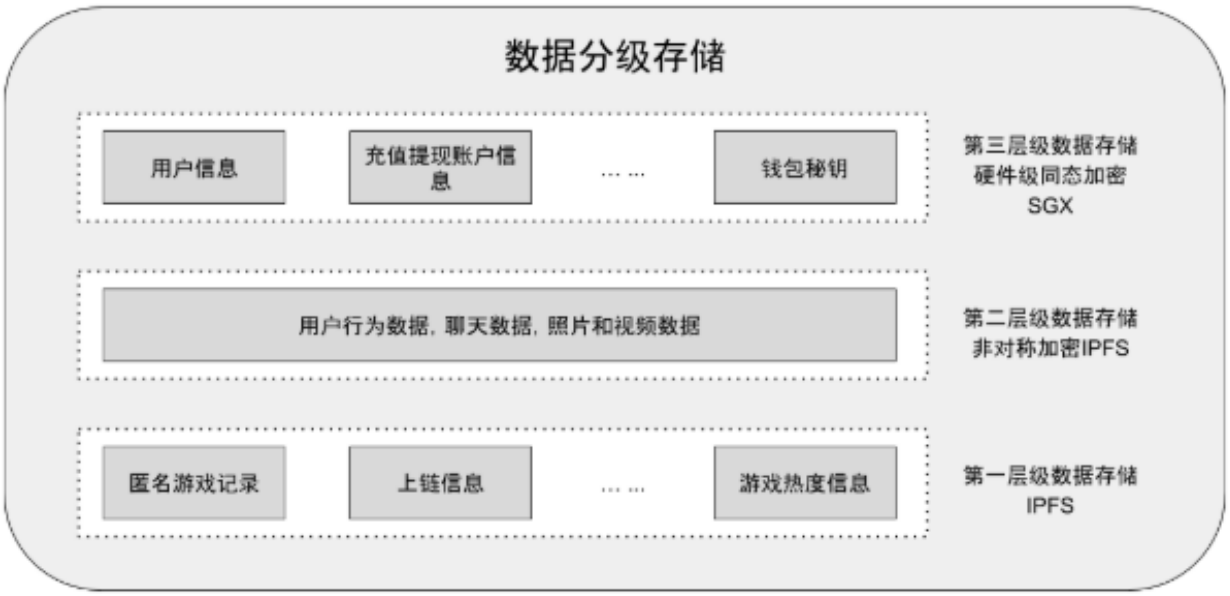
图：块树说明图（赋值）

检查点高度的计算方法和合理的链举例 r→b1→b2→b3

如果验证人违反了任何上述规则，则违规的证据可以作为交易包含在区块链中，此时验证人的全部存款将被支付给证据提交者作为“发现者费用”。在目前的Unicorn平台中，停止执行违规操作需要对Unicorn平台的TPoS提议者进行51%的攻击。

2.3 分级数据管理

Unicorn认为数据安全非常重要，它们按照私密等级，被分为上图中的三类。Unicorn也按照这个分类层级，对它们进行相应的加密操作，用来保证用户的数据安全。用户对他们的数据有完全的掌控权，可以自主分享，修改和删除自己的数据。具体来说，Unicorn认为，用户可以给服务提供商分享敏感度低的行为数据从而得到更好的定制化服务，用户可以跟朋友加密分享较为敏感的信息建立交际圈，同时，像区块链钱包秘钥这样的数据需要用最先进的技术加密，并且不应该与任何人分享。上图正式从这三个层级将数据分级存储在Unicorn平台上。下面我们将具体介绍这三个数据层的实现技术。

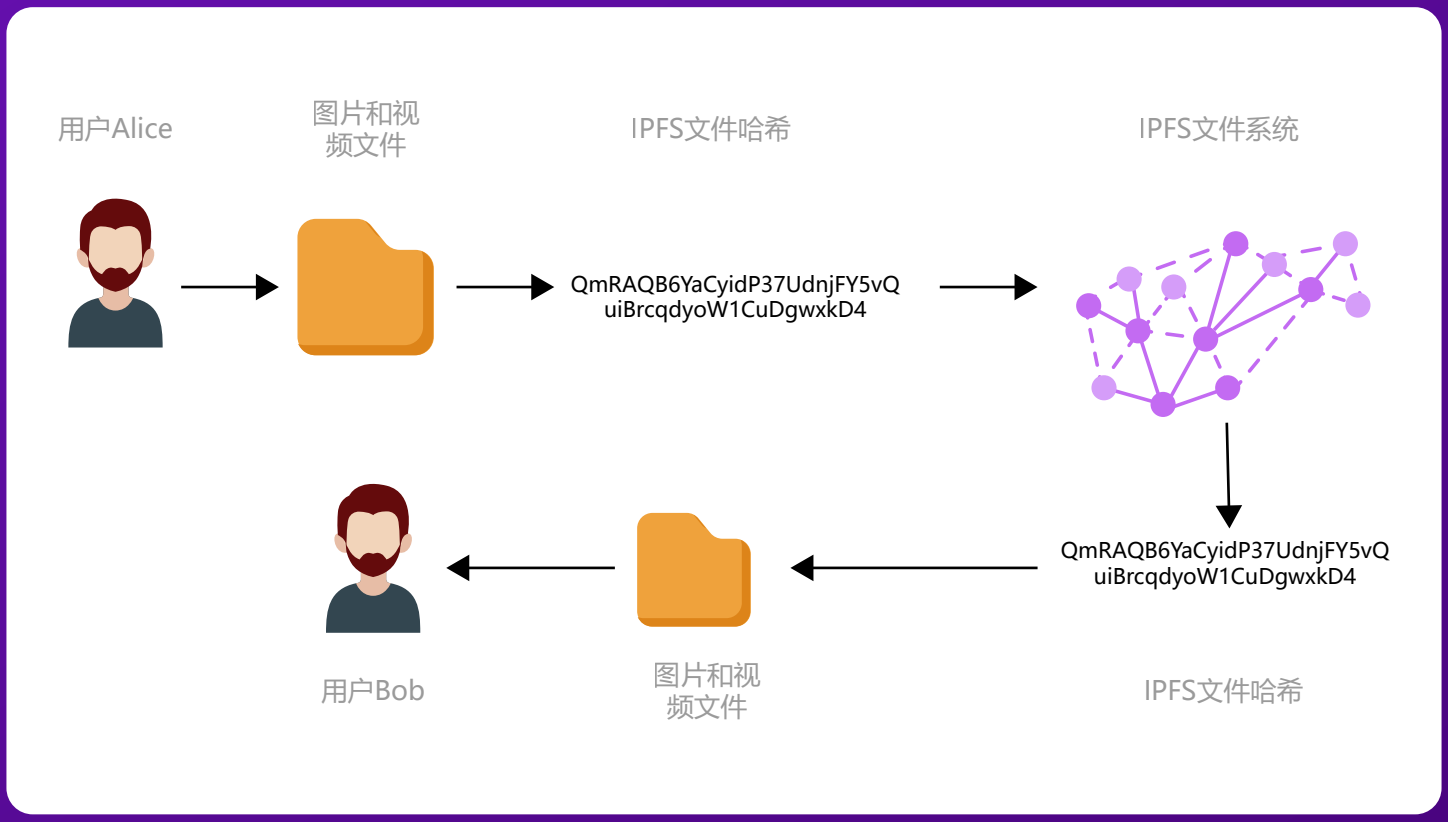


2.3.1 第一层级数据存储 -- IPFS

星际文件系统（InterPlanetary File System），即IPFS，由Protocol Labs的人员创建。它是一种点对点协议，每个节点都存储一组散列文件。想要检索任何文件的客户端可以访问一个抽象层，这个抽象层允许客户端使用它想要访问文件的哈希值来找到该文件，然后IPFS将通过节点路由向客户端提供该文件。

IPFS可以视为是与BitTorrent类似的存储协议。它使用完全去中心化的交互方式，提供通过哈希引用来对文件进行多种操作，从而实现更丰富的程序化交互。

下图是IPFS工作流程的简化示意图



图：IPFS工作流程示意图

1.用户Alice想要将图片或者视频文件上传到IPFS；

2.她将他想上传的图片或者视频文件文件放在她的工作目录中；

3.她告诉IPFS他想要添加这个文件，IPFS会生成该文件的哈希值，并上传至IPFS分布式存储网络；

4.该文件在IPFS网络上存储成功。

那么，现在假设用户Alice希望通过IPFS与他的朋友用户Bob来分享这个文件。他只是告诉用户Bob上面第3步由IPFS生成的文件的哈希值，然后用户Bob就可以通过该哈希值在IPFS网络中找到该文件了。

像比特币这样的区块链，会有一个专门的BPM模块，这个BPM模块可以非常高效的存储简单的文本记录，这就是加密货币非常适合在区块链上运行的原因。 在加密货币的应用场景中，BPM这个模块只需要记录交易的发送方，收款方和加密货币的数量，这使得BPM模块能非常高效的执行该任务。但是，如果需要存储大量的其他数据，比如文本数据或个人信息数据，区块链上的存储效率是非常糟糕的，这是因为每一次生产块，都需要计算和验证所有这些数据的哈希以保持链的完整性，这会导致块的生产非常没有效率。

图：区块中IPFS的文件Hash的存在形式

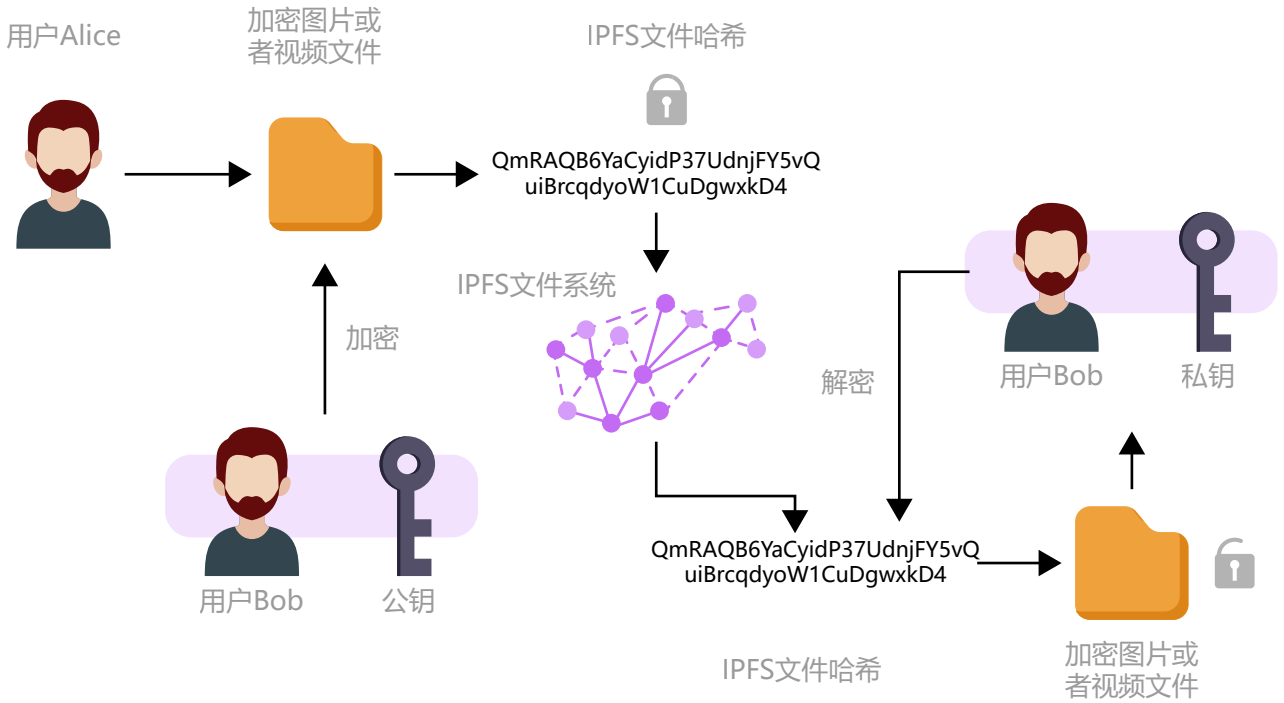
这时就需要如上图，使用IPFS与区块链结合，来解决这个问题。 Unicorn只在区块链上存储由IPFS生成的需要存储文件的哈希值，而不是上面的BPM。这保证了区块链所需的数据简单性，同时，我们也获得了IPFS分布式文件存储完全去中心化的特性。在下面的章节，我们会讲到如果对IPFS存储的文件进行加密，这保证了Unicorn使用IPFS存储较为敏感的文件的安全性。

2.3.2 第二层级数据存储 -- 非对称加密IPFS

这一部分数据，Unicorn采用非对称加密算法[1]对他们进行加密，加密密钥是由Unicorn根据用户ID和生成密钥的时间随机生成的，密钥被密钥密码和手机验证保护，密钥和用户账户相关联，只有能够访问特定账户，并且能够验证密钥密码和手机验证的用户，才能通过使用密钥解密查看这一层级的数据。

Unicorn可以在用户授权的情况下和特定的用户分享较为敏感的数据。这些数据将使用用户的密钥对，对数据进行加密，然后将加密后的数据存储在IPFS上。

非对称加密允许Unicorn用户使用想要分享数据的用户的公钥对数据进行加密，然后特定的用户就能够使用他们的密钥对这些文件进行解密。没有被授权的用户，无法对该文件进行解密，因为他们的密钥和加密数据所使用得公钥无法匹配。



图：使用IPFS存储非对称加密数据

上图中展示了在用户同意的情况下，有选择性的对他们的数据进行分享。具体来说：

1.用户Alice想要分享一个图片或者视频文件文件，但只允许用户Bob访问；

2.Unicorn将这个图片或者视频文件文件使用用户Bob的公钥对其进行加密；

3.Unicorn将这个加密后的图片或者视频文件文件上传到IPFS，得到相应加密文件的哈希值；

4.用户Bob可以检索并解密文件，因为他拥有用于加密文件的公钥的关联私钥，其他用户无法解密该文件，因为他们缺少用户Bob的私钥。

2.3.3 第三层级数据存储 -- 硬件级同态加密SGX

这一层级的数据被保存在Unicorn超级节点共同维护的加密服务器上（提供加密服务器是加入Unicorn超级节点的一个必要条件），不会被存储在去中心化的IPFS网络存储中，为了保证数据的绝对安全。

Unicorn对于这一层级的数据进行最严格的加密，因为这一层级的数据一般为区块链钱包秘钥，不应该和第三方分享，但是为了提供无感化跨链服务，Unicorn在跨链时，需要访问这些秘钥。众所周知，同态加密是满足这个愿景的最佳选择，但是这项技术还不成熟，它要么限制了可以对数据集执行的操作类型，要么会导致系统性能的显著下降。因此，我们选择采用更先进的加密技术，即SGX加密技术[2]。SGX加密允许Unicorn在数据处理期间，也保护用户数据的隐私性，

SGX: 即英特尔软件保护扩展，是一组安全相关的指令代码，内置于一些现代英特尔中央处理器（CPU）中。 它们允许用户级和操作系统代码定义内存的私有区域，称为安全区，其内容受到保护，无法由安全区外部的任何进程读取或保存，包括以更高权限级别运行的进程。

同时能够支持在加密下的任何类型的数据操作。简单来说，Unicorn可以定义数据处理方法，然后通过Unicorn的数据中心，在加密的状态下，对数据进行操作和建模。这都是通过硬件级SGX同态加密实现[3][4]。

硬件加密允许Unicorn区块链平台为用户数据构建最先进和安全的存储。具体来说，SGX代表Intel Software Guard Extensions。顾名思义，它是英特尔系统（IA）的扩展，用于增强软件安全性。此方法无法识别和隔离平台上的所有恶意软件。相反，它将合法软件的安全性封装在飞地中并保护其免受恶意软件的侵害。特权或非特权软件无法访问该飞地。也就是说，一旦软件和数据在

Enclave[enclave：芯片上的安全飞地。支持云计算的芯片也有类似技术，英特尔、ARM和AMD都有，名为可信执行环境。AMD有其安全执行环境(SEE)，ARM的是可信区域(TrustZone)，英特尔是软件保护扩展(SGX)]中，即使操作系统和VM管理程序也不会影响Enclave中的代码和数据。 Enclave的安全边界仅包含CPU和自身。 SGX创建的Enclave也可以理解为可信执行环境（TEE）。 SGX中的CPU可以并行运行许多安全区域，Unicorn区块链平台中间件支持这些区域。使用Unicorn区块链平台 SGX技术，系统通过切换CPU的硬件模式进入可信模式，仅使用必要的硬件形成完全隔离的特权模式，加载微型微内核操作系统以支持任务调度，并完成身份认证。通过使用Unicorn区块链平台 SGX技术，将Enclave构建为完全隔离的特权模式的具体实现方案如下：

1.将虚拟机映像载入磁盘。

2.为加密的应用程序代码和数据生成密钥证书。 Unicorn区块链平台 SGX技术提供了更先进的密钥加密方法。其密钥由SGX版本密钥，CPU密钥和Unicorn区块链平台分配给用户。新密钥是在密钥生成算法下生成的。此密钥用于加密要加载的应用程序的代码和数据。

3.首先将应用程序的代码和数据加载到SGX Loader中。 SGX装载机准备将其装入Enclave。

4.在Unicorn区块链平台 SGX可信模式下动态构建Enclave。

5.要加载的程序和数据首先由密钥证书以EPC（Enclave Page Cache）的形式解密。

6.SGX指令用于证明解密的程序和数据是可信的并加载到Enclave中，然后复制加载到Enclave中的每个EPC内容。

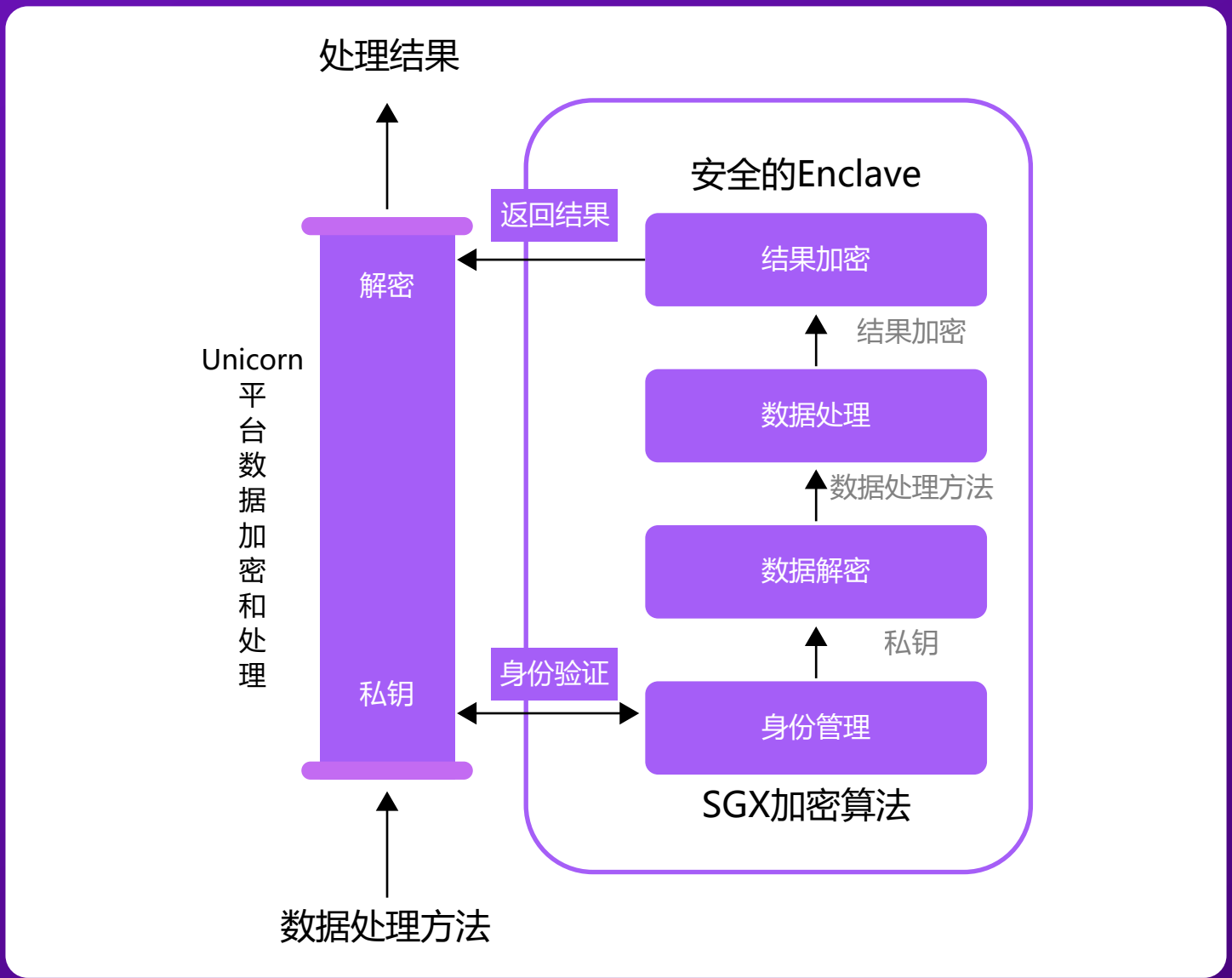
7.由于使用了硬件隔离，Enclave的机密性和完整性得到进一步保证，确保不会发生不同Enclaves之间的冲突，并且它们不允许相互访问。

8.启动Enclave初始化程序，禁止继续加载和验证EPC，生成Enclave凭据，加密凭证，并将其作为Enclave徽标存储在Enclave的TCS（线程控制结构）中，以恢复和验证其身份。 。

9.完成SGX的隔离，硬件隔离的Enclave中的镜像程序开始执行，基于SGX技术的硬件隔离完成。

enclave：芯片上的安全飞地。支持云计算的芯片也有类似技术，英特尔、ARM和AMD都有，名为可信执行环境。AMD有其安全执行环境(SEE)，ARM的是可信区域(TrustZone)，英特尔是软件保护扩展(SGX)。

为了访问SGX Enclave，Unicorn区块链平台 SGX认证算法首先确定是否激活了Enclave模式，然后确定访问请求是否来自Enclave。 如果是肯定的，则认证算法继续判断，如果不是，则返回访问失败。 在生成Enclave之前基于凭证，它们用于验证访问请求是否源自相同的Enclave。 如果是，则授予访问权限。 否则，迭代Enclave的身份凭证记录表，替换下一个Enclave凭证以匹配，直到测试所有Enclaves凭据。 如果匹配失败，Unicorn区块链平台 SGX身份验证算法将返回访问失败。



上图说明了在安全的Enclave中处理用户数据的功能。 首先，用户的密钥由Unicorn区块链平台中间件和Enclave验证。 然后，包围区在安全的Enclave环境中解密所请求的数据。 数据解密后，Enclave使用数据处理函数处理数据，例如跨链操作。 最后，处理结果被加密并返回到Unicorn区块链平台中间件，该中间件解密数据后将结果返回给数据使用者，即Unicorn平台。

2.3.4 同态加密算法

我们强调CPA安全性不会阻止攻击者篡改加密的消息，例如，将消息x的加密更改为x的加密，其最后一位被翻转。同态加密将这种情况发挥到极致，实际上要求在保持CPA安全性的同时不允许任意方式篡改加密。

定义

我们说如果存在算法NAND，则对于每个 (e,d)←G(1n)，a,b∈{0，1} 并且 a*←Ee(a)，b←Ee(b)，具有一位消息的CPA安全公钥加密方案(G,E,D)是完全同态的。

NANDe(a*,b*) ≈ Ee(aNANDb)

其中 ≈ 表示统计不可区分性（即,n-ω(1)统计距离），并且aNANDb表示¬(a∧b)。

我们强调算法NAND不会将密钥作为输入。否则，它将太过简单：只需解密a*，b*，计算aNANDb并重新加密。

NAND的普适性

很简单地表明每个日志门都可以使用少量NAND表示，因此获得以下声明：如果(G,E,D)是同态加密，那么每个都有一个算法EVAL，对于（e ,d)←G(1n)，x1,...,xm∈{0,1}，如果x*i = Ee(xi)，C是映射{0,1}m到{0,1}的布尔电路，然后

EVALe(C,x1,...,xm) ≈ |C|μ(n)Ee(C(x1,...,xn))

我们说 D ≈ ε D'，如果它们的统计距离最多为ε，则μ是一些可忽略不计的函数，并且|C|表示C的门数。特别地，如果C是多项式大小，那么这两个分布在统计上是不可区分的。

基本协议

我们将逐步描述协议及其安全性。

公共输入：布尔电路C：{0,1}n → {0,1}。
证明者的私人输入：x∈{0,1}n，使得C(x)= 1。
步骤1：证明者运行（e,d）← G(1n)，将e发送给验证者。
步骤2：证明者将 x* = Ee(x1)···Ee(xn)，发送给验证者。
Step3：证明者计算 c* = EVAL(C,x*)，发送c*给验证者。
步骤4：证明者将 d = Dd(c*) 发送给验证者。如果 d = 1，验证者接受。

完全零知识

上述协议尚不清楚该协议对于恶意验证者也是零知识。问题是这样的验证者可能会使用步骤3来获取解密框的任意查询，而这是他自己无法做到的事情，因此至少违反了零知识的精神。为了解决这个问题，我们稍微修改协议：

步骤4：证明者只向 d 发送一个承诺（例如f(x)，r，⟨x,r⟩ ⊕ d，其中 f 是单向置换）。

步骤5：验证者发送它在产生步骤3的密文时使用的所有随机性。证明者确认情况确实如此，否则中止。

步骤6：证明者发送 d 以及用于产生承诺的随机性。

这可以被证明可以保持健全性，因为即使对于计算无界的证明者来说，健全性仍然存在，并且承诺方案是完美的约束力。另一方面，我们现在可以拥有一个使用倒带的模拟器，在回答问题之前提取验证者的选择。

Unicorn差分隐私保护模型

对于一个有限域Z，$z \in Z$为Z中的元素，从Z中抽样所得z的集合组成数据集D，其样本量为n,属性的个数为维度d。对数据集D的各种映射函数被定义为查询，用

$F = \{f1, f2, \ldots \ldots\}$

来表示一组查询,算法M对查询F的结果进行处理使之满足隐私保护的条件，此过程称为隐私保护机制。设数据集D和D',具有相同的属性结构,两者的对称差记作DΔD'，|DΔD'|表示DΔD'中记录的数量。若|DΔD'|=1，则称D和D'为邻近数据集。设有随机算法M，PM为M所有可能的输出构成的集合.对于任意两个邻近数据集D和D'以及PM的任何子集SM，若算法M满足

$Pr[M(D) \in S6] \leq \exp(\varepsilon) \times Pr[M(D) \in SM]$

则称算法M提供ε差分隐私保护,参数ε称为隐私保护预算。

隐私保护预算ε用来控制算法M在两个邻近数据集上获得相同输出的概率比值，它事实上体现了M所能够提供的隐私保护水平。在实际应用中，ε通常取很小的值，例如0.01、0.1或者ln2，ln3等。ε

越小表示隐私保护水平越高。当ε等于0时，保护水平达到最高，此时对于任意邻近数据集，算法都将输出两个概率分布完全相同的结果，这些结果也不能反映任何关于数据集的有用的信息。因此，ε的取值要结合具体需求来达到输出结果的安全性与可用性的平衡。

差分隐私保护可以通过在查询函数的返回值中加入适量的干扰噪声来实现。加入噪声过多会影响结果的可用性，过少则无法提供足够的安全保障。因此，敏感度是决定加入噪声量大小的关键参数，它指删除数据集中任一记录对查询结果造成的最大改变。在Unicorn中差分隐私保护中包含了两种敏感度，即全局敏感度和局部敏感度。

全局敏感度，设有函数$f: D \rightarrow R$d，输入为一数据集D，输出为d维实数向量。对于任意的邻近数据集D和D'，

$$GSf = \max D,D' ||f(D) - f(D')||$$

称为函数 f 的全局敏感度，其中$||f(D) - f(D')||$为 1 阶范数。

局部敏感度，设有函数$f: D \rightarrow R$d，输入为一数据集 D，输出为 d 维实数向量。对于给定数据集 D 和它任意近数据集 D'，

$$LSf = \max D' ||f(D) - f(D')||$$

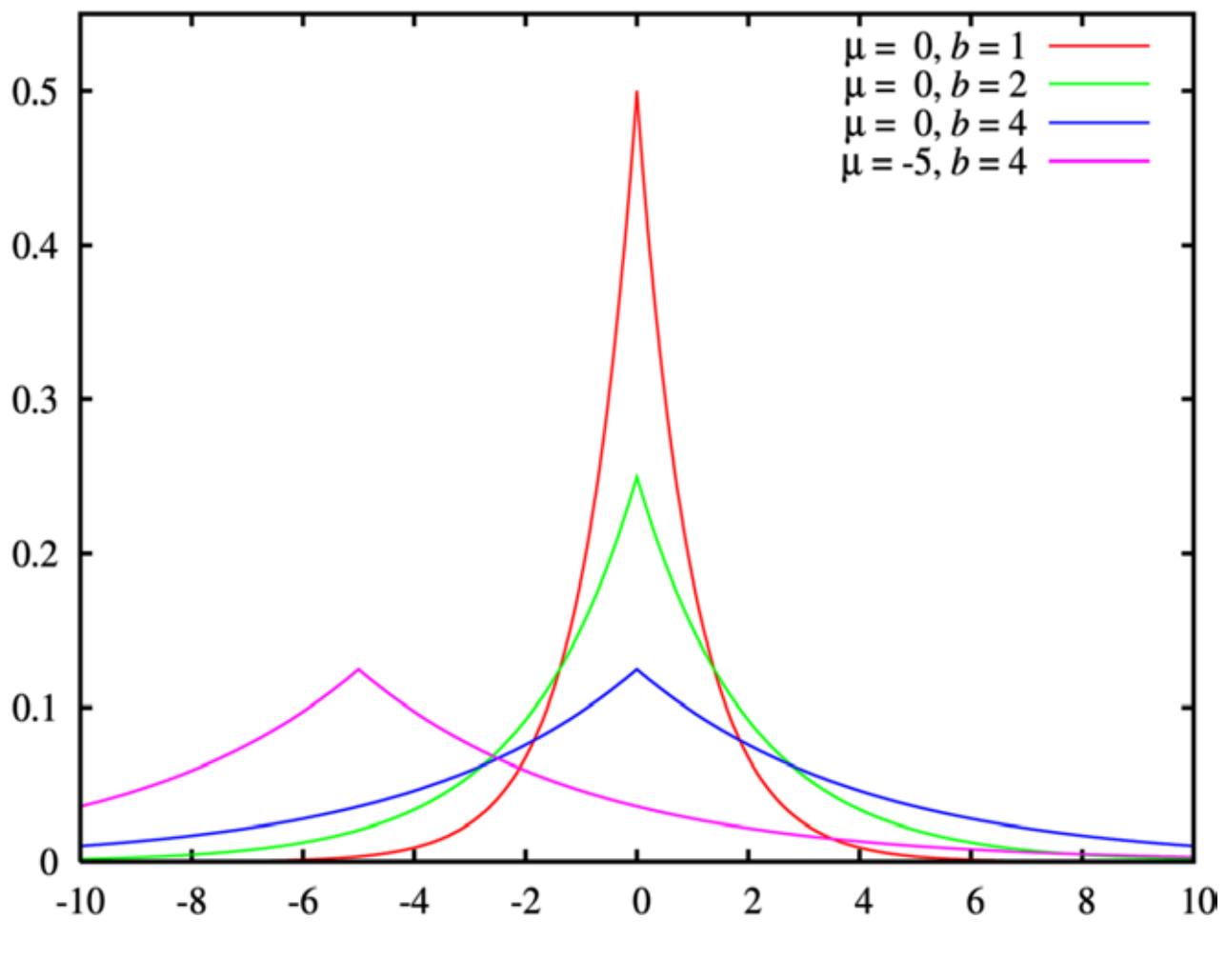
称为函数 f 的在 D 上的局敏感度。

Unicorn中，既包含了对于数值型结果又包含了非数值型的结果，因此Unicorn同步引入了拉普拉斯 机制与指数机制对跨链数据进行隐私保护。

拉普拉斯机制，通过向确切的查询结果中加入服从 拉普拉斯 分布的随机噪声来实现ε差分隐私保护。记位置参数为 0、尺度参数为b的 拉普拉斯 分布为 Lap（b）,那么其概率密度函数为

$$P(x) = (1/2b)\ \exp\ (-|x|/b)$$

给定数据集 D，设有函数f: $D \rightarrow R$d,其敏感度为Δf，那么随机算法$M(D) = f(D) + Y$提供ε差分隐私保护，其中Y ~ Lap($\Delta f/\varepsilon$)为随机噪声，服从尺度参数为$\Delta f/\varepsilon$的 拉普拉斯 分布。从不同b值的拉普拉斯分布可以看出，ε越小，引入噪声越大。



指数机制，设随机算法 M 输入为数据集 D，输出为一实体对象r ∈ Ranger，q(D，r)为可用性函数，Δq为函数q(D，r)的敏感度，若算法 M 以正比于exp(εq(D,r)/2Δq) 的概率从 Range 中选择并输出 r，那么算法 M 提供ε差分隐私保护。

2.4 跨链钱包

现阶段，市面上有大量良莠不齐的数字货币钱包存在，而不少开发团队在以业务优先的原则下，暂时对自身钱包产品的安全性并未做到足够的防护，一旦出现安全性问题会导致大量用户出现账户货币被盗，而由于数字货币实现的特殊性，被盗资产非常难以追回，因此钱包的安全性是至关重要的。因此Unicorn使用硬件同态加密存储用户账户信息，即使是Unicorn在使用用户私钥信息来实现币币兑换和跨链服务时，也是在硬件加密的环境下完成的。对于硬件同态加密，我们将在下文中详细介绍其实现架构。同时Unicorn跨链钱包还实现了PIN保护和加密货币以及在设备中加密货币的12字种子。Unicorn永远无法访问种子，因此它们在你的设备上受到保护。

Unicorn跨链钱包通过RPC计算机通信协议实现。该协议允许运行于一台计算机的程序调用另一台计算机的子程序，而程序员无需额外地为这个交互作用编程。RPC的主要功能目标是让构建分布 式计算（应用）更容易，在提供强大的远程调用能力时不损失本地调用的语义简洁性。Unicorn钱包通过RPC接口调用来和各个区块链网络进行交互，达到能同时管理多个币种的目的。

具体来说，RPC（远程过程调用）是一种计算机通信协议，它允许在一台计算机上运行的程序调用另一台计算机的子程序，而程序员不必编程该交互。RPC是一种分布式计算模式。客户端向服务器发送请求以执行多个进程。服务器接受请求并使用客户端提供的参数完成计算。计算完成后，结果返回给客户端。有很多RPC协议，比如最早的CORBA，Java RMI，Web服务的RPC风格，Hessian，Thrift，和REST API。

下图展示了一个典型的RPC过程。它可以被看作以下步骤：

1.客户端在本地调用RPC调用函数；

2.客户端序列化服务接收到调用后，负责将方法和参数组装成可在网络传输的消息。

3.客户端序列化服务会查找服务器地址并将消息发送到服务器。

4.服务器序列化服务接收到消息后解码消息。

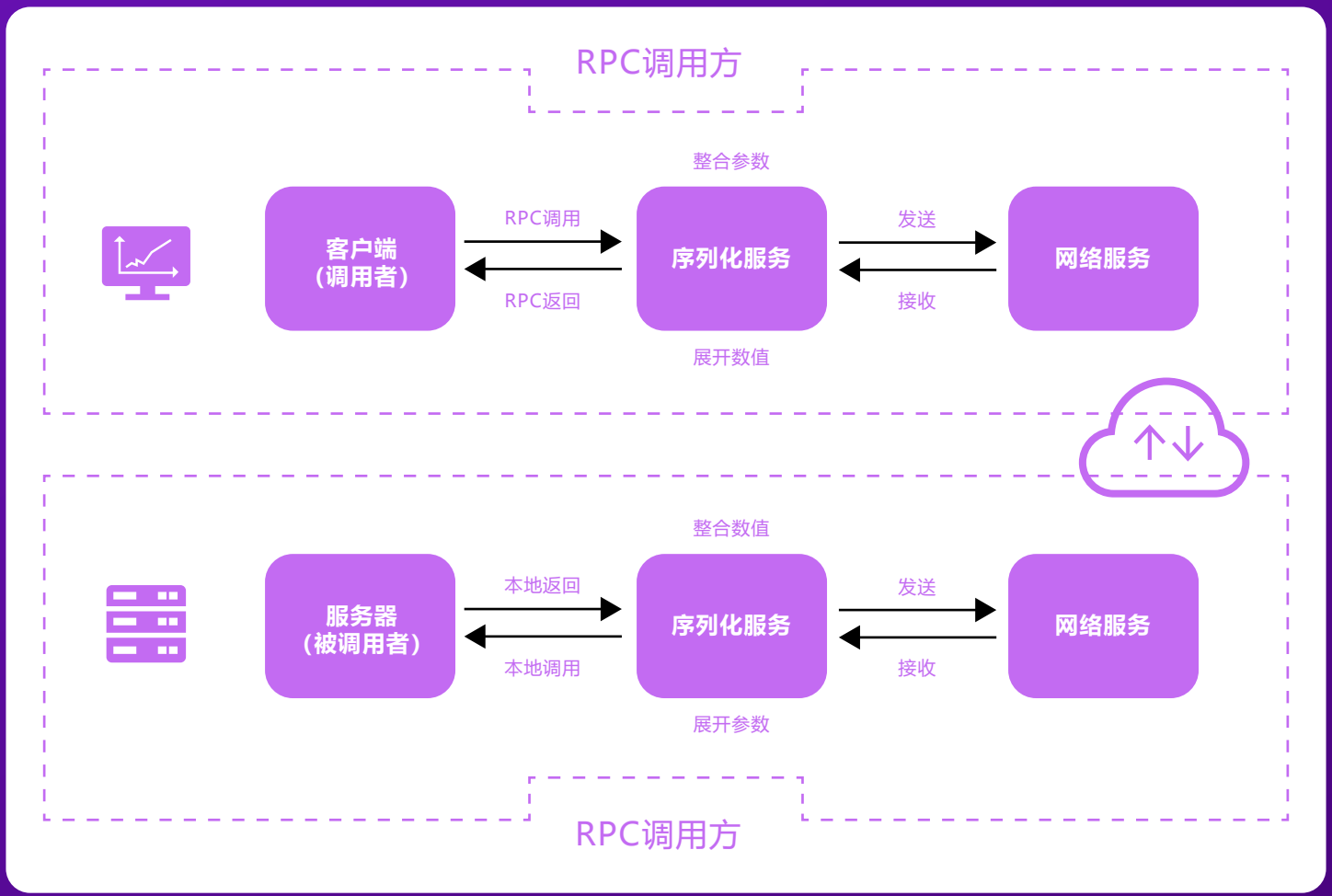
5.服务器序列化服务根据解码结果调用本地服务。

6.服务器将执行结果返回到服务器序列化服务；

7.服务器序列化服务将返回的结果打包成消息发送给客户端。

8.客户端序列化服务接收消息并对其解码。

9.客户得到最终结果。



图：RPC调用示意图

通过使用RPC，Unicorn可以让用户在Unicorn账号上管理多个区块链资产，并且可以进行实时币币兑换和游戏消费。具体RPC接口，我们将在跨链平台接口中详细讲解。存储在Unicorn钱包中的用户私钥是绝对安全的，Unicorn使用硬件同态加密技术对用户私钥信息进行严格加密，即使是Unicorn在使用用户私钥进行一些系统操作时，也无法在非加密状态下导出秘钥信息。硬件同态加密的技术实现我们将在下一章中详细介绍。

3. 跨链接口平台 Phoenix

Unicorn跨链平台Phoenix的架构由三层组成，每层实现不同的功能：

1.资产层：分布式存储和管理来自各种外部区块链的数字资产，为数字资产提供跨链交易操作；

2.图层：交易和智能合约的执行和存储；

3.协议层：用于存储和跨链执行分布式应用的必要基础设施。

3.1 资产层

安全的数字资产（Safe Digital Assets）

对于使用不同的区块链协议，跨链操作层将使用数字资产安全（SDA）概念存储。 这是在跨链交互协议（Cross chain Interaction Protocol）下运行的数字资产的分布式存储。 该存储解决方案可以安全地存储加密货币和数字资产，并进行跨链交换。 跨链操作层同时是Unicorn基础设施和其他网络上的区块链节点的一部分。 可以将这些节点与集中式加密货币交换网络进行比较，每个交换网络支持不同区块链上的一个或多个专用（本地）钱包，并根据一组规则执行跨链操作。

1.dStorage - Unicorn用户的文件存储，包括分布式应用程序的归档。

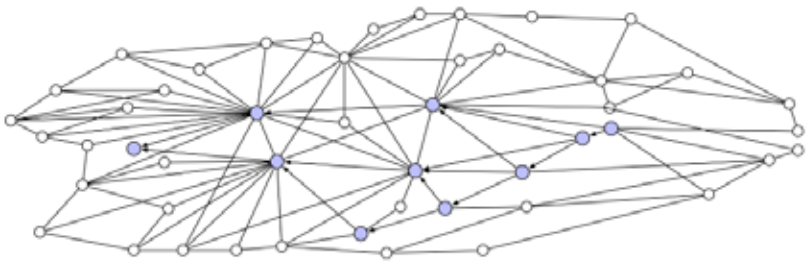
2.dDNS - 存储Unicorn服务和Web项目地址的分布式分类帐。来自注册表的信息只能使用项目所有者的私钥或网络公共投票机制删除。

3.dCloud - 用于用户节点的分布式分布式云平台。云使网络参与者可以运行自己的节点并通过API和UI访问它们。

网桥 - 与Unicorn集成的第三方区块链平台上的节点

3.2 图层 -- 排序化DAG

排序化DAG是一种有向无环图，它将节点的评级考虑在内。该算法是准确定的异步拜占庭容错（aBFT）协议，其具有基于关于在计算一致性时向网络传播信息的评级的领导者的选择。该算法使用网络状态的存储器和事务传播历史。排序化DAG算法是水平可扩展的，因此随着网络参与者数量的增加，交易数量也将与Unicorn的增长成正比。交易处理时间是一个重要因素。在初步测试中，它的范围为1到30秒。该算法在不受信任的环境中工作，条件是至少三分之二的网络参与者广播事务是可信的。排序化DAG不使用Proofof-Work，并且不会形成通常意义上的区块链。这意味着所有生产力都用作在网络内传输数据。同时，交易确认依赖于对发起人的历史的反向分析，这使得该系统与第一代区块链相比更有效。



图：Unicorn网络中DAG示意图

3.3 跨链协议层 CIP

跨链交互协议（Cross chain Interaction Protocol）是一种协议，该架构基于共识算法（RAFT

+TPoS）和数字资产安全（SDA）算法。 CIP的任务是在网络上执行跨链事务。当涉及的资产在两个或多个不同的区块链网络上定位/运行时，跨链交互协议（CIP）是在分布环境中交换数字资产的过程之间的交互的原始版本。

3.3.1 Unicorn跨链交易节点

由于没有挖矿机制，Unicorn上的所有交易都没有挖矿产生的佣金。但是参与交易的节点通过下列过程获得佣金：

1.节点通过签名来确认交易。随着它的签名，该节点在RAFT+TPoS协议下规定了报酬。

2.一旦交易被网络接受，该节点就会自动获得报酬。

3.节点的赌注被解锁，并且由于伪造而未被阻挡的代币数量增加 - 新代币的数量与节点的评级成正比。

4.如果节点确认更诚实的交易并提出更大的押注，节点将获得更多的利润。

下面是Unicorn参与跨链的节点类型及其功能：

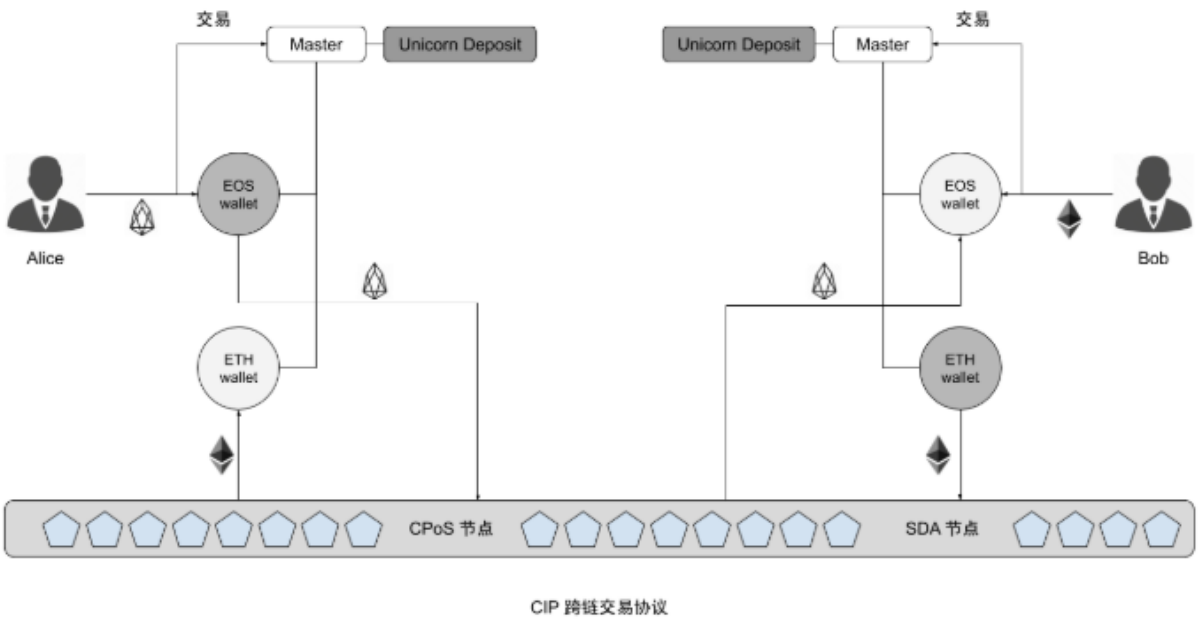
1.普通节点：组合成一个扁平的点对点网络，确保平台的功能和分布化。

2.完整节点：提供查看完整交易历史的功能，额外的平台保护。

3.主节点：支持运行SDA层和应用层算法的应用层子网的功能。

4.桥：将事务传输到外部区块链网络。

5.服务节点：执行应用任务以确保网络功能。



CIP 跨链交易协议

3.3.2 CIP算法中跨链事务模型的描述

外部区块链网络（比特币，以太坊等）通过网桥连接到网络的应用层，网桥既是外部区块链中的节点，也是Unicorn的一部分。

用于跨链交互协议（CIP）算法允许以三种不同方式执行数字资产的跨链交换：

1.直接通过安全数字资产（SDA）类型的数字资产存储。

2.通过多重签名钱包。此模型称为通过多重签名钱包进行交换。

3.通过保险金来承保用户进行交易的意图。这种模式被称为通过保险交换（电子保险）。

跨链交互协议（CIP）中使用的每种技术都代表了一种独立的业务逻辑形式，并为Unicorn的运行提供了不同的技术可能性和功能。不同技术的结合使得可以完成各种支付任务并执行广泛的不同财务操作。跨链交互协议（CIP）将拥有一个开放的API，允许第三方开发人员在外部服务和应用程序中使用Unicorn技术。

3.3.3 保证权力下放的机制

Unicorn有一种机制，用于在节点组合成池时保证分布。 当决定在参与者之间分配来自锻造的代币时，排序化DAG考虑了分布程度。

如果网络参与者的数量减少，但是RAFT+TPoS算法激活的代币的生产率和总数量增加，那么奖励的分配以这样的方式变化，即节点的收入对其评级的依赖性相对于其数量减少。 因此，排序化DAG确保将节点组合到池中没有经济动机。

Unicorn上的交易不仅仅是将数字资产从一个用户转移到另一个用户的问题。其功能将扩展到创建网络以允许多种不同的操作，包括：

1.在网络中传输加密货币和数字资产。

2.在网络参与者之间形成和转移数字资产组合。

3.涉及创建，编辑和删除用户私钥的操作。

4.在网络内投票。

5.创建和编辑属于用户的智能合约和数据。

6.创建指向dApp和其他数据的链接

从经典理解交易的角度来看，资产从一个用户转移到另一个用户，Unicorn上的交易有两种类型：

1.内部，其中有关交易的信息仅记录在Unicorn区块链中。这些包括：

●在用户之间免费转移Unicorn加密货币；

●跨链交换，数据记录在外部区块链（比特币，以太坊等）。

2.外部，其中关于交易的数据记录在外部区块链中，例如，当用户资金（BTC，ETH等）被导入Unicorn或从Unicorn导出时。

3.3.4 跨链智能合约

Unicorn的核心目的是支持与外部区块链平台的互动，这意味着不仅可以管理交易，还可以管理其他区块链网络上的智能合约。

跨链智能合约将在各种区块链平台上定义智能合约的请求层次和执行条件，充当路由器。Unicorn可以在图层中存储外部区块链平台的智能合约，并使用Bridges作为外部系统的联系点来执行它们。在外部平台上执行智能合约的佣金只会在外部区块链上收取。没有计划在Unicorn平台内收取佣金。该平台还将为外部区块链平台提供标准和经过测试的智能合约（模板）库，可通过API使用该模块创建轻型或全智能合约。

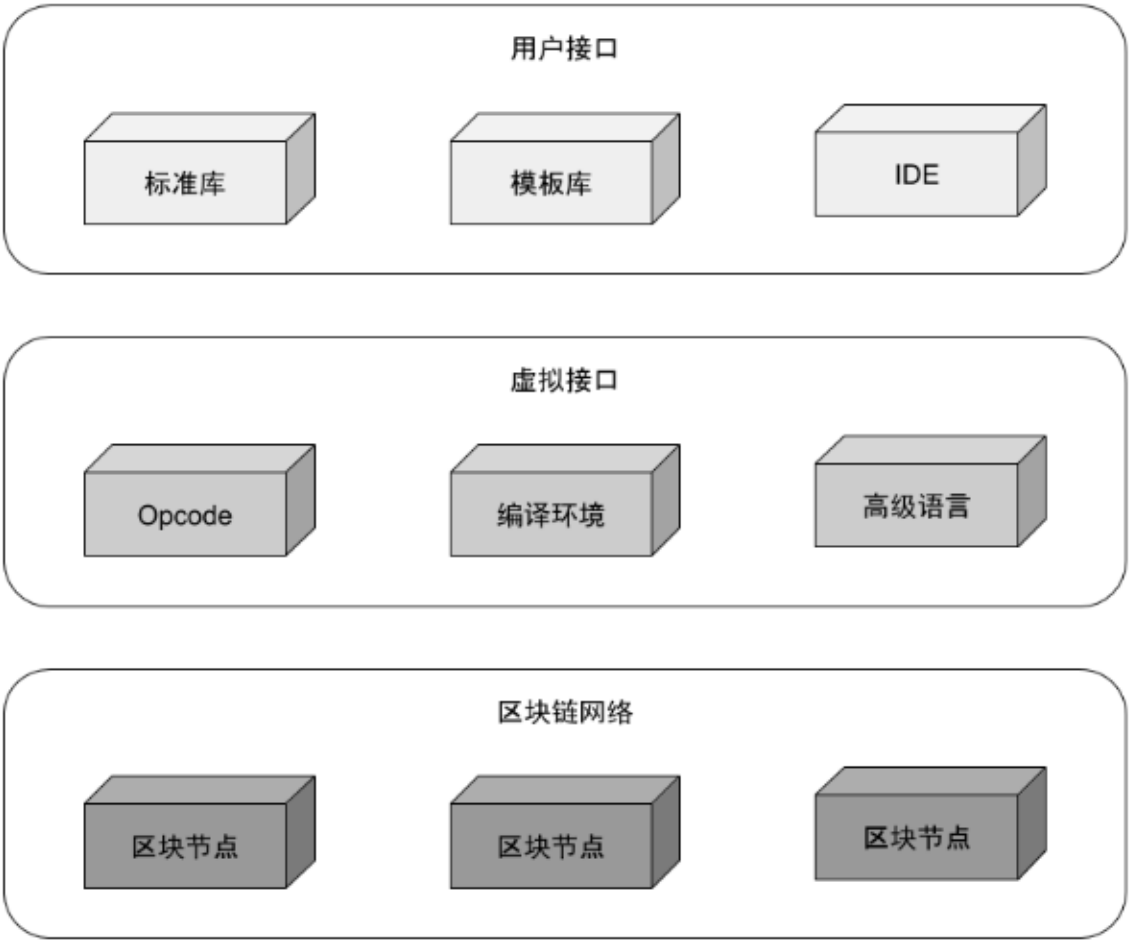
同时，项目团队还在开发Unicorn Framework，这是一个开放的组件库和基础架构，适用于分布式应用程序（SDK）的开发人员，允许他们使用平台的所有功能和Unicorn的资源实现成熟的分布式应用程序。网络参与者，应用程序实现任何业务逻辑或与他们自己的分布网络。

如何保证结果的一致性和真实性是一个去中心化系统所面临的最大问题。中心化系统的解决方案是完全依赖一个中心化节点，操控各种数据操作，例如博彩或公开融资的结果。但中心化的解决方案容易受到恶意攻击，一个节点的宕机就会引起整个系统的下线。性能方面，中心化节点的吞吐量有限，

容易成为整个系统的瓶颈，在大部分情况下都不能满足全球化用户的需求。

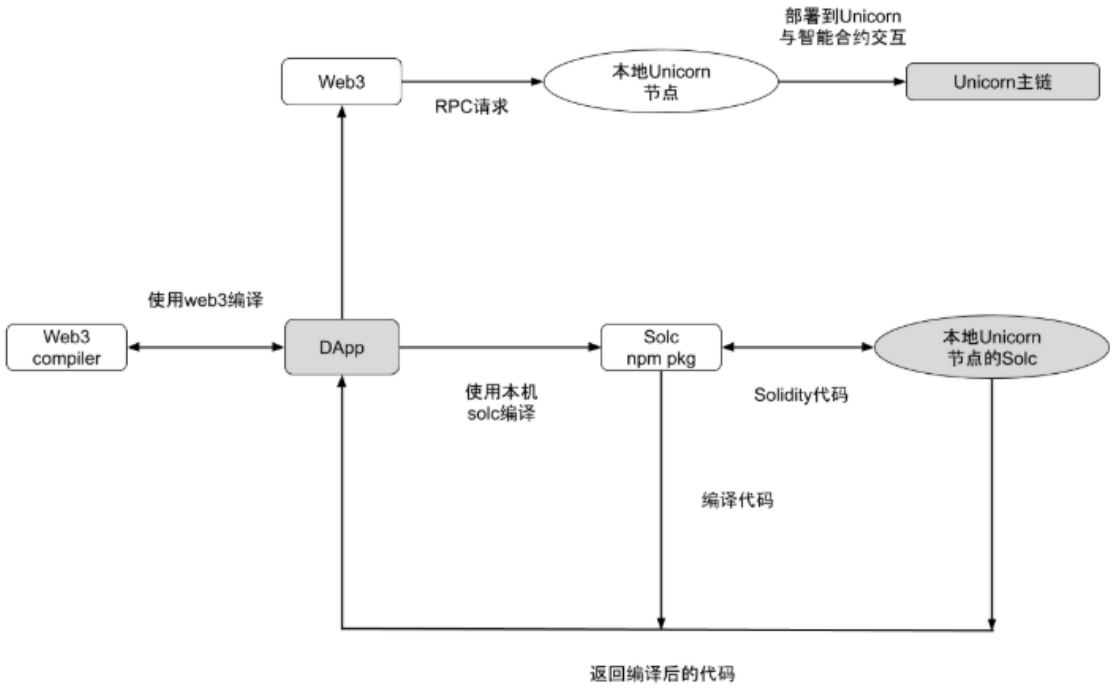
以博彩应用举例，Unicorn系统采用区块链智能合约，将博彩过程部署到区块链上进行，使得任何人都可以验证博彩结果的真实性，博彩奖金发放的公平性。并且，一旦智能合约部署，任何人在博彩前或者博彩后，都不能对智能合约所约定的内容做任何的修改，使得整个博彩过程公正公开，奖金的分配由智能合约自动完成——在博彩结果确定后自动分配Unicorn到赢得博彩的用户登记的钱包地址。更进一步说，智能合约的部署，防止任何人对博彩的规则和结果在博彩的任何阶段做任何修改。这是因为智能合约按照区块链的规则去中心化的部署在全球任意多个匿名的计算机上，这些计算机通过RAFT+TPoS算法达成共识，即没有任何一个人能篡改部署在区块链上的智能合约和其执行结果，除非有一个人或者机构能够掌握全球超过50%的RAFT+TPoS超级节点，这显然是不现实的，所以Unicorn系统是基于区块链的安全性上建立的完全安全透明可信的一个生态系统。

为了向用户提供更好的智能合约开发环境，Unicorn开发了一套更完备的智能合约平台，实现了名为Solidity的图灵完备的智能合约编程语言，并提供虚拟机来支持在Unicorn区块链进行智能合约的开发、测试和部署，具体智能合约架构如下图所示：



图： 智能合约模块图

为了将智能合约部署到分布在全球的计算机上执行，我们需要通过Web3或者本机的solc编译环境编译智能合约，然后通过RPC请求将合约部署到Unicorn的主网上，最后在全球各个计算机上执行并达成共识。Unicorn系统提供友好的人机交互界面，一方面帮助用户查看部署在Unicorn主网的智能合约内容，另一方面帮助项目方或其他机构编写新的智能合约。



图： 智能合约执行的示意图

4. 大数据智能平台

下图展示了Unicorn大数据平台的模块架构，下面我们将对其中的重要模块进行详细介绍和说明。

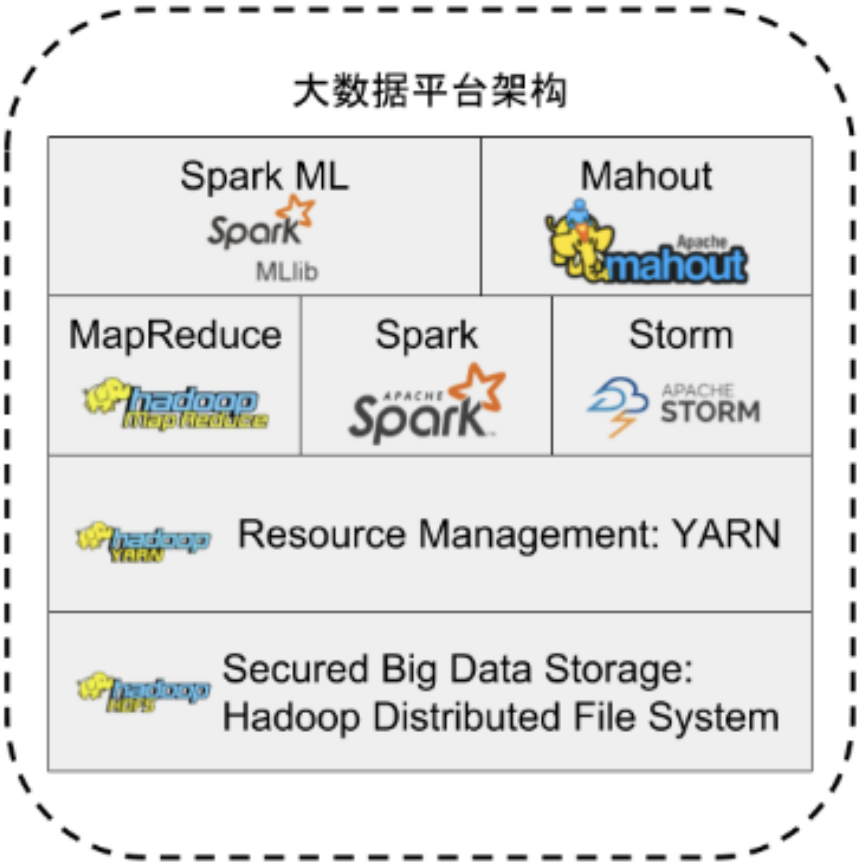


图. Unicorn大数据平台模块图

4.1 分布式存储：HDFS

为了存储大量的玩家数据和游戏记录，Unicorn利用Hadoop分布式文件系统（HDFS），这是

大数据存储基础架构中最先进的技术。具体来说，HDFS是一个Apache Software Foundation项目，它是Apache Hadoop项目的子项目[5]。 Hadoop非常适合存储大数据，并使用HDFS作为其存储系统。 HDFS允许我们连接多个计算机或群集中包含的节点（通常是虚拟机），并在这些计算机上分发数据文件。然后，我们可以将数据文件作为无缝文件系统进行访问和存储。对数据文件的访问以数据流方式处理，这意味着应用程序或命令直接通过MapReduce处理模型执行。此外，HDFS具有容错能力，可以对大型数据集进行高吞吐量访问。因此，HDFS是我们的大型数据集（即玩家数据和游戏记录）的理想分布式存储系统。

HDFS与其他分布式文件系统有许多相似之处，但存在一些差异。一个明显的区别是HDFS的“一次写入多次读取”模型，它降低了并发控制要求，简化了数据聚合，并支持高吞吐量访问。这也是Unicorn选择使用HDFS的主要原因，因为玩家数据和游戏记录是不可变的，即它们是只读数据记录。 HDFS的另一个特性是将数据处理逻辑迁移到数据，这样做比将数据迁移到数据处理逻辑更有效率。这在我们的使用场景中至关重要，因为我们允许数据用户使用Unicorn计算资源来处理他们的自定义数据处理功能。它显然节省了数据迁移成本。 Unicorn实现了HDFS的许多目标。以下是一些最基本的内容：

- 通过检测故障并应用快速自动恢复来实现容错
- 简单可靠的聚合模型
- 处理逻辑接近数据，而非接近处理逻辑的数据
- 跨异构硬件和操作系统的可移植性
- 可靠的存储和处理大量数据可扩展性
- 通过跨多个计算机集群分发数据和处理来节省成本
- 通过将分布式数据和逻辑并行处理到数据所在的多个节点来提高效率
- 通过自动维护多个数据副本并在发生故障时自动重新部署处理逻辑来实现可靠性

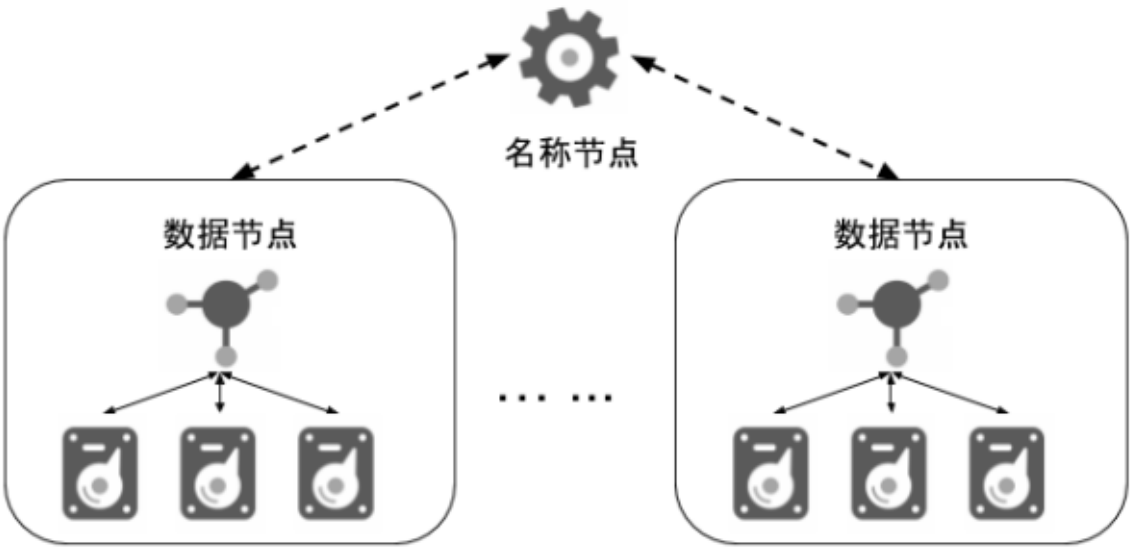


图. Unicorn大数据平台HDFS结构图

4.2 群集资源管理器：YARN

Unicorn使用YARN管理和仲裁Unicorn平台资源。它试图最大限度地利用平台资源，同时满足数据用户的计算配额。换句话说，它会自动将用户的计算任务分配给最适合执行的服务器。具体来说，在YARN体系结构中，全局ResourceManager作为主要后台进程运行，该进程通常运行在独立的一台计算机上，仲裁每个应用程序可以获得的群集资源。ResourceManager跟踪群集中有多少可用节点和可用资源，并且协调用户提交的应用程序，并帮助它们获取这些资源。ResourceManager是拥有此信息的唯一进程，因此它可以以共享，安全，多租户的方式（例如，基于应用程序优先级，队列容量，ACL，数据位置等）进行分配（或调度）决策。 此外，ResourceManagers不关心应用程

序或任务的类型。所有应用程序框架特定的代码都被传输到它的ApplicationMaster，以便任何分布式框架都可以被YARN支持——只要有人为它实现了相应的ApplicationMaster。因此，归功于这种通用方法，YARN能够管理许多不同工作负载的集群资源。在Unicorn平台中，YARN监督MapReduce，Storm，Spark，SparkML，Mahout等的运行。这种方法显然提供了许多优点，包括：

1.更高的集群利用率，一个框架中未使用的资源可以被另一个框架使用

2.降低运营成本，因为只有一个“一体化”集群需要管理和运行

3.减少数据移动，无需在Hadoop YARN和运行在不同机器群集上的系统之间移动数据

4.管理单个集群还将导致更环保的数据处理解决方案。它使用更少的数据平台空间，使用更少的硅浪费，更少的功耗和更少的碳排放，仅仅因为我们在更小但更高效的Unicorn集群上执行了相同的计算量。

4.3 大数据处理库：Spark

Unicorn使用Spark[6]搭建了大数据处理平台，Spark非常适合构建大规模，低延迟的大数据分析和学习的应用程序。这正是Unicorn平台所需要的。用户可以使用该平台，在取得用户授权的情况下，对用户的游戏数据进行数据挖掘和机器学习任务。具体来说，Spark是一个类似于Hadoop的开源集群计算环境，但两者之间存在一些差异。这些差异使Spark在某些工作负载中表现更好。特别是，Spark可以在内存中启用分布式数据集，除了提供交互式查询外，还可以优化迭代工作负载。

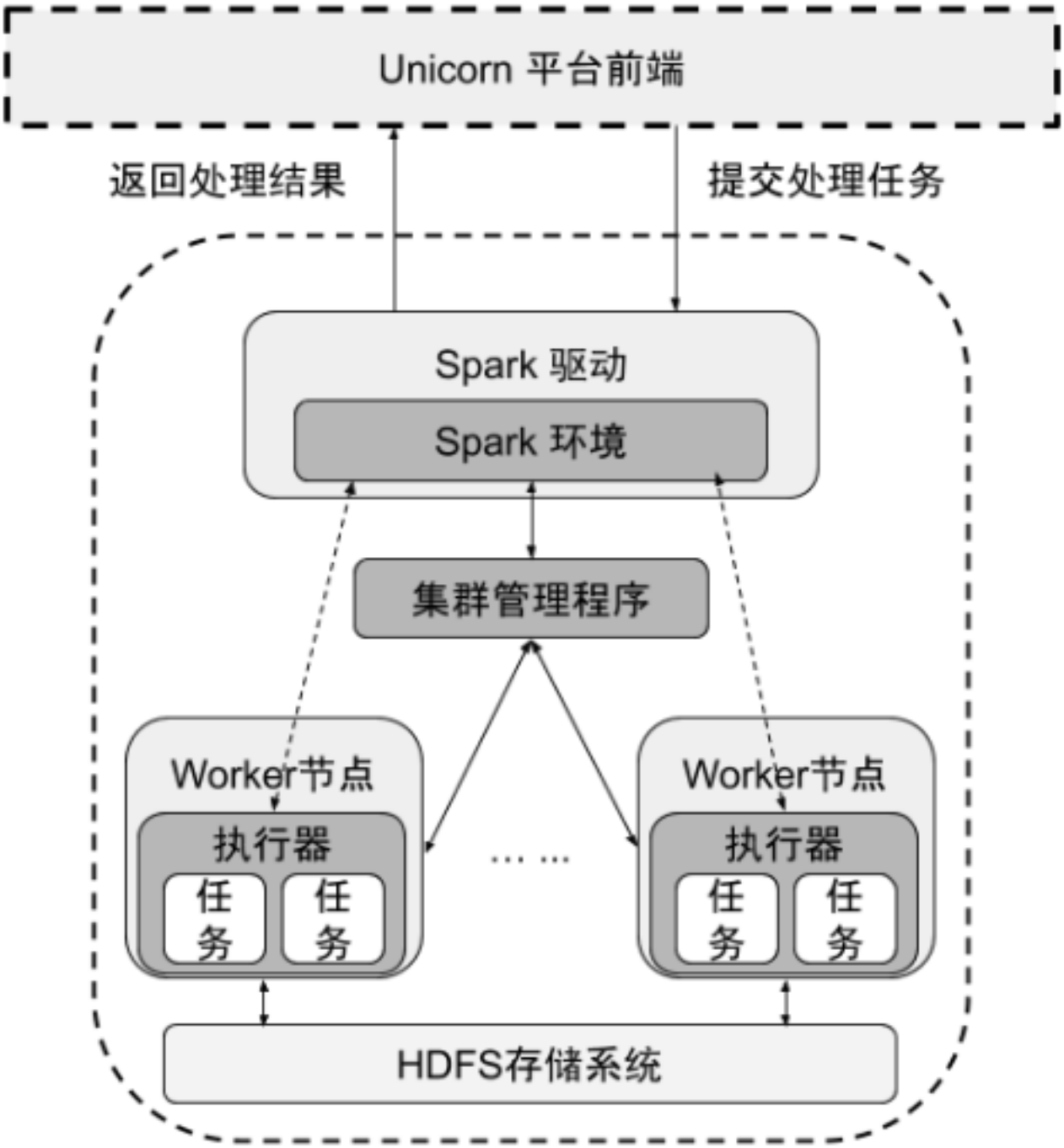


图. Spark大数据处理平台

4.4 流处理库：Storm

Unicorn在其大数据平台上使用Storm[7]进行流处理。 Storm能够处理来自任何数据源的连续数据流。 Unicorn在很多场景中都使用Storm，包括通过监控平台和Unicorn区块链的运行记录，实时的分析生态的运行状态。

Storm是一个免费的开源分布式实时计算系统。通过Storm，可以非常轻松地处理无限的数据流。像Hadoop批处理大数据一样，Storm可以实时处理数据。Storm很简单，可以使用任何编程语言。在Storm之前，做实时处理是一件痛苦的事情：需要维护一堆消息队列和消费者，他们形成了一个非常复杂的图形结构。做实时处理是非常痛苦的，我们主要的时间花在关注发送消息的位置，接收消息的位置以及如何序列化消息。真正的商业逻辑只占源代码的一小部分。

Storm彻底解决了这些问题。它适用于分布式场景，抽象消息传递，并自动处理群集计算机上的流计算，使用户能够专注于实时业务处理逻辑。Storm实现了一个数据流模型，在该模型中数据流经许多转换实体所形成的网络。Tuple就像是一个数据结构，可以表示标准数据类型，如int，float和byte数组或用户定义的类型。每个Tuple由一个唯一的ID标识，可用于为流拓扑中的各种组件构建数据源。

如下图所示，水龙头表示数据流的来源。一旦水龙头打开，数据将通过闪电连续处理。图中有三个流，每个数据流包含连续的数据Tuple，它们携带特定的数据。Tuple通过流过闪电中不同的转换实体进行处理。

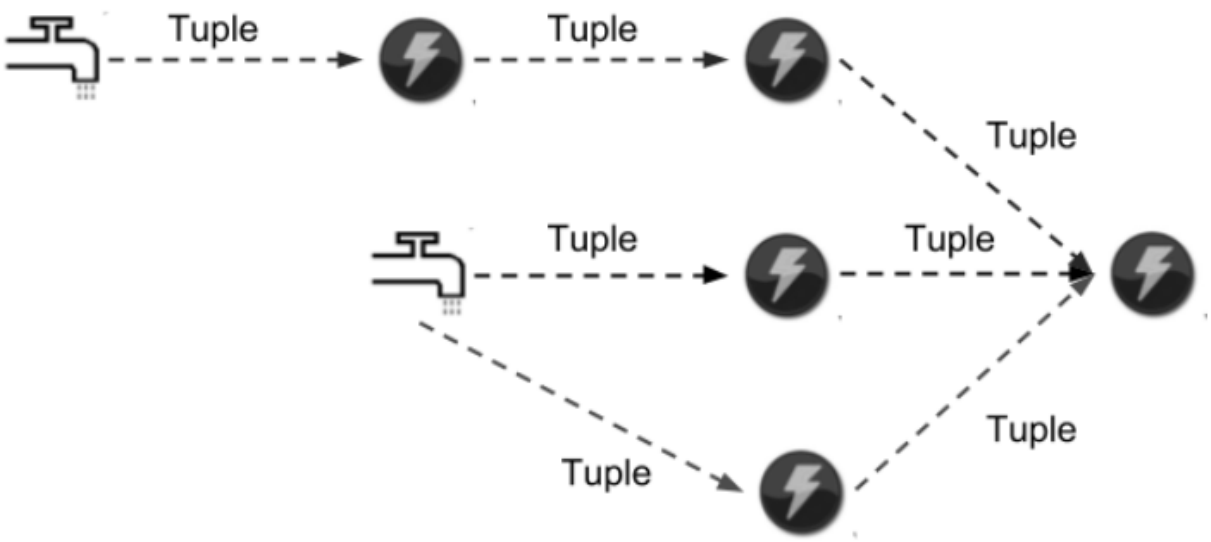


图. Storm流处理示意图

Storm不会对数据输入源和输出数据的目的地施加任何限制。像Hadoop一样，有必要将数据放入它自己的文件系统HDFS中。在Storm中，只要用户实现相应的代码来读/写（反序列化/序列化）数据，就可以使用任何数据输入源和任何数据输出。在典型场景中，输入/输出数据基于消息队列（如Kafka或ActiveMQ），但数据库，文件系统或Web服务也是可能的数据源。

4.5 机器学习库：Tensorflow

我们使用最先进的由Google开源的机器学习类库Tensorflow[8]对Unicorn的机器学习应用进行开发。Tensorflow和Spark很像，都是将运算定义在图上，执行程序的过程就是遍历这个执行图，然后由图上的各个节点对数据进行各种运算。具体来说，Unicorn提供Tensorflow运行环境，数据使用者提交到Unicorn的机器学习代码都可以部署到Unicorn数据中心，对用户公开的或授权的数据进行机器学习和数据挖掘。

5. 参考文献

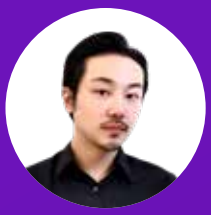
[1] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM 21, 2 (February 1978), 120-126. DOI=http://dx.doi.org/10.1145/359340.359342

[2] COSTAN, V., AND DEVADAS, S. Intel SGX explained. Technical Report., Cryptology ePrint Archive, Report 2016/086, 2016.

[3] Dario Catalano and Dario Fiore. 2015. Using Linearly-Homomorphic Encryption to Evaluate Degree-2 Functions on Encrypted Data. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). ACM, New York, NY, USA, 1518-1529. DOI: https://-doi.org/10.1145/2810103.2813624

[4] Brickell E.F. (1990) A Survey of Hardware Implementations of RSA. In: Brassard G. (eds) Advances in Cryptology — CRYPTO’ 89 Proceedings. CRYPTO 1989. Lecture Notes in Computer Science, vol 435. Springer, New York, NY

6. 团队



吴国晋(Sunny Ng)

UNI 创始人及CEO

大学先后从英国诺丁汉大学获得经济学学士、英国华威大学获得金融会计硕士，拥有多年的海外工作和连续创业经验。15年创立XBLOCK在挖矿市场打拼，旗下有算力平台，以及矿池，矿池拥有全球前十五ZCASH及以太币算力，17年获4000万种子轮融资并成立区块链软件公司，积累了多年区块链底层开发技术以及多条公链上的 DApp 开发经验。

https://www.linkedin.com/in/sunny-ng-6523b049/

Onn Mah

UNI CTO

香港理理工工大大学电子子计算学系学士士 • 以太坊Dapps游戏Hogsmash.io 及 KittyPillar.com的创始人人 • VR 遊戏 VR Alien Blaster 开发总监 • 16年从事电脑程序开发的经验。

魏国章（William Wei）

UNI 顾问

1965年2月24日出生于台湾，美籍华人，软件、移动互联网领域专家，现任南京核果信息科技有限公司CEO，创毅科技集团 CTO 、赛伯乐投资集团合伙人 .

拥有美国麻州大学电脑工程和北卡Kenan-Flagler商学院工商管理双硕士学位。1993年被乔布斯亲自招聘，成为NeXT公司唯一的华人核心软件成员。1997年跟随乔布斯重新回归苹果。2008年在苹果企业解决方案部门研发MDM/MAM。

7.募资情况

共进行三轮融资，第一轮为无币天使融资共100BTC，第二轮私募融资共50万U，第三轮公募融资共10万U，应社区，市场要求，为项目初期投资者及UNI粉丝社群的公平起见，为共建可持续发展生态，于第三轮公募前退回第二轮私募机构融额的90%，即第二轮融资5万U

8.Token分配：总额100亿Token

下面是图