Core 芯算链白皮书

1 简介

基于权益共识机制与基于工作量证明的智能合约相比有明显的优势，而智能合约的商用化对智能合约平台也有更严苛的要求，如向后兼容性、可利用简单支付验证协议的轻钱包进行跨组织信息流交互。现有的智能合约平台——以太坊使用了代价高昂的基于工作量证明共识机制，且需要下载全链数据，此外还有未来多次分叉的可能性，这些因素直接限制了以太坊智能合约平台的广泛应用。以太坊缺乏形式语义学也是一个安全问题。本白皮书提出 Core 芯算链智能合约平台，并提供便于行业部署且符合行业最佳实践的智能合约模板库，解决了智能合约无法大规模商用的问题，以此同时也提高社会技术应用适用性。白皮书中还讨论了芯算链采用抽象账户层支持不同的虚拟机的优势，并计划开发 x86 架构的虚拟机，为智能合约的应用提供了更多的选择。

在区块链系统中，智能合约是指合约参与方通过协商沟通、并利用共识机制进行各方行为约束的协议。智能合约可以应用在诸多行业，如金融科技，物联网和数字签名等。最初智能合约验证交易依靠所谓的工作量证明机制 PoW。智能合约的核心技术是分布式账本，也就是我们常常提到的区块链。借助区块链技术，可以无需信任中心化的机构而进行交易的全记录。随着区块链技术的逐渐普及，名为比特币的点对点数字货币和支付系统以及一系列基于协议层的相关操作也逐渐为大众所知 。比特币利用工作量证明机制进行交易验证需要耗费大量的算力、电力资源，并且大批量的交易处理效率低下，需要超过一小时才能进行交易的确认。

与比特币不同，智能合约系统支持图灵完备语言，使用 Solidity 一种语法类似 JavaScript 的高级语言，特别针对用以太坊虚拟机（Ethereum Virtual Machine）编写的智能合约而设计的。作为目前领先的智能合约平台，以太坊在设计上仍有一些缺陷。首先，以太坊使用工作量证明机制进行交易验证，较差的可拓展性导致其无法进行大规模商业应用；其次， 由于缺乏正式验证手段引发的安全漏洞使基于 Solidity 编写的智能合约被黑客入侵，导致该以太坊项目损失约 6000 万美元，并引起以太坊硬分叉 。之后拒绝式服务攻击导致了以太坊的又一次硬分叉 ，可以预见 随着基于权益共识机制和区块分片处理引入以太坊，可能会导致更多的硬分叉出现。

除了上述提到的原因，还有其他因素也制约了以太坊大规模商用。如无法进行自动化的跨组织信息流交互、对内部隐私合约和外部合约缺乏区分化的隐私保护机制、缺乏安全稳定的进行基于权益共识机制交易验证的区块链虚拟机、缺乏可供验证的智能合约编程语言、缺乏无需下载全链数据的轻钱包应用、缺乏支持简单支付验证协议的智能合约移动端解决方案 。以太坊虚拟机还经常遭到异常处理攻击和诸如针对交易排序和时间戳等依赖性攻击。智能合约系统也希望通过采用侧链和未花费交易输出（UTXO）来实现行业可扩展性，并实现与比特币或其他或彩色币的兼容性。此外，比特币闪电网络的双向微支付通道也为行业可扩展性提供了更多的可能 。

账户抽象层（Account Abstraction Layer, AAL）是 Core 连接 UTXO 流通层和智能合约平台层的关键，也是芯算链实现的一大创新。通过 AAL 可以将 UTXO 模型转换成可供不同虚拟机执行的账户模型，比如以太坊 EVM，或是 x86 虚拟机，并且可以把虚拟机的帐户余额通过 AAL 转换为 UTXO。这实现了余额和智能合约平台的分层设计，流动性采用比特币的 UTXO，并且原生支持多重签名算法，作为余额的记账、转账更具有安全性；而由于智能合约平台是图灵完备的，很难避免编程出现 bug，把流通层和和智能合约平台分离比以太坊更具优势。

为了使区块链的智能合约平台能力得到进一步的扩展，Core 计划实现的 x86 高性能虚拟机可以直接通过 C、C++、rust、go 等语言进行智能合约的编程，也可以支持其他可以在 x86 架构上运行的编程语言。x86 架构虚拟机可以解决以太坊 EVM 中一些问题，比如不支持标准库、生成的字节码过大、不支持浮点、难以调试等。

区块链的治理是区块链可持续发展的重要议题，对区块链的发展有重要影响，而建立更有效的区块链网络治理模式，是区块链行业未来发展中面临的巨大难题和挑战。比如如何更有效的解决比特币网络扩容的问题，如何在比特币不同的扩容方案中做出大多数都认可的选择，以及如何达成这种共识，从而避免 1MB 区块还是 2MB 区块长达 2-3 年的争论？至今为止比特币网络对这个问题还在不停的争论中。如何达成分叉的共识，并降低分叉对生态系统的影响是整个区块链行业都需要思考的问题。为此 Core 芯算链通过设计分布式自治协议 (Decentralized Governance Protocol, DGP) 为区块链网络提供了一种新的治理模式。
智能合约本质是一段代码，理论上采用了图灵完备的智能合约可以实现任意复杂度的协议设计，甚至是区块链的核心协议，如共识部分的代码等；权衡效率、安全等；安全性考虑， Core 的 DGP 协议仅适用于在安全范围内对特定参数进行更改，同时对参数生效时间采取一定的时间限制。通过内嵌到创世区块的智能合约来治理区块链网络的参数，并实现一个去中心化的网络自治机制，实现区块链网络的自动升级和快速迭代，而不用担心软硬分叉对网络和社区带来的影响。通过 DGP 协议和机制，区块链网络在分叉或者升级过程中，用户无需对核心钱包升级，即可对某些分叉实现自适应，最大程度地减少对区块链网络、生态系统以及用户产生的影响。可管理的每个区块链网络参数都由独立的 DGP 智能合约控制， DGP 治理以投票的方式完成对区块链网络参数修改提案的决策，或通过投票对管理和治理席位进行增加或删除。当提案通过，提案会更新到相应的 DGP 合约变量保存，钱包和区块链上的所有节点可以定时检查 RLP 储存从而判断是否需要更改到新的区块链网络参数。
对于大多数的加密货币来说，区块生产者会收到区块奖励和交易费用。交易费用主要目的：

1)激励区块生产者收录交易进入区块；
2)阻止区块链网络上的垃圾交易。

基于 PoW 共识算法的比特币交易费用设计的非常有效，因为其为了产生区块，需要支付挖矿设备和电力费用。然而在 PoS 共识算法下，需要花费的设备和电力费用并不大，如果某个挖矿参与者拥有大量的币，只需要等待一定的时间就可以生成区块，虽然生成区块的概率会随网络难度波动，但并不为 0。这意味着在某些情形下，特别是在支持图灵完备的智能合约的环境下，有可能会以比较小的代价用于垃圾交易，进而攻击基于 PoS 的网络。Core 因此对 PoS 3.0 的奖励机制做了一个修改，用于部分解决通过垃圾交易攻击而产生的无用 stake 问题，这种新的 PoS 机制命名为 MPoS (Mutualized Proof Of Stake)。

2 Core 芯算链性能优势

Core 芯算链的主要目标之一是建立基于权益共识机制（PoS）、并支持 UTXO 的智能合约模型。由于选择了权益共识机制，PoS 的含义是区块创造者的选择是通过持有的数字货币量来决定的。在 PoS 中，通过锻造（forge）或铸造（mint）的方式，而不是通过挖矿获取相应的数字货币，并且没有块奖励，矿工仅收取一定的交易费用。Core 芯算链兼容比特币 UTXO 模型和符合 AAL 规范的虚拟机，其中第一个兼容的虚拟机是以太坊的 EVM，后续将实现 x86 架构的虚拟机，支持实现多种流行的智能合约编程语言。本着追求务实的设计方法，Core 芯算链全面支持“面向移动端”策略以扩大商业应用可能性。“面向移动端”策略使 Core 芯

算链能够将区块链科技带给更多的互联网用户，并逐渐将 PoS 交易验证去中心化。

2.1 UTXO 模型与账户模型

在未花费交易输出（UTXO）模型中，交易使用未花费的比特币作为输入，此时输入的 UTXO 就会作废，而输出是另一个 UTXO, 比特币数量上变化的结果会返还到发送者。一定数量的比特币在不同私钥持有人之间进行转移，新的未花费交易输出在交易中花费，并记录在区块上。在比特币交易中，UTXO 可用交易接收方公钥地址生成的秘钥进行解锁。需要说明的是，矿工在 coinbase 交易中生成比特币，这个过程中并没有包含任何输入。同时，比特币利用脚本语言只能进行有限的操作，并以堆栈（分为主堆栈和 Alt 堆栈）的形式进行数据处理，并遵循“后进先出”（LIFO）原则。开发者在比特币客户端定义了五种交易标准，分别为：P2PKH（Pay to Public Key Hash）、 P2PK（Pay to Public Key）、多重签名（少于 15 个私钥签名）、P2SH（Pay to Script Hash）和 OP_RETURN。利用这五种交易标准，比特币客户端可以满足复杂的支付逻辑。

但比特币脚本语言并不是图灵完备的，无法实现循环功能。这极大地制约了交易执行量和交易复杂度。此外，比特币脚本语言作为编程语言并没有被广泛使用。当然，这些限制也降低了诸如无限循环在内的复杂支付逻辑安全漏洞的风险。当然 UTXO 模型也有诸多优势：任何人可以通过比特币公共账本对每一笔交易历史进行查询；UTXO 有良好的可拓展性，能够同时处理多个地址发起的交易请求。此外，UTXO 模型也提供了隐私保护，用户可以使用变更地址作为 UTXO 输出。但 UTXO 并不提供状态信息，因此 Core 芯算链的目标是在 UTXO 模型的基础上加入全新设计，提供全新的智能合约平台。

与 UTXO 模型不同，以太坊使用了账户（Account）模型。具体来说，以太坊通过账户状态的改变进行价值和信息的交换与传输，并通过长度为 20 字节的随机数作为指针以确保交易处理的唯一性。用于支付交易费用、供内部使用的加密货币称为以太坊。合约代码是可选的，而账号的存储默认为空。以太坊账户有两种类型，一种由外部私钥控制的外部账户，另一种由合约代码控制的合约账户。外部账户用于信息传输的创建、交易签名。合约账户用于收到内部存储读写操作信息后创建合约或发送其他信息。以太坊中的账户余额管理与日常生活中的银行账户管理相类似。每一个新产生的区块都有可能影响其他账户的全局状态。每个账户都有各自的余额、存储和代码空间用于调用其他账户或地址，并存储相应的代码执行结果。现有的以太坊账户系统中，用户通过客户端远程调用合约账号进行 P2P 交易。尽管通过智能合约向多个账户发送信息是可以实现的，但具体交易信息只有参与交易的账户可见，无法在以太坊公共账簿上进行追踪仍无法实现。

综上所述，我们认为以太坊在扩展性方面的瓶颈使比特币 UTXO 模型拥有更多的优势。UTXO 模型与 Core 芯算链提供的平台一致性更相关，因此 Core 芯算链决定使用 UTXO 模型作为交易模型的基础。

2.2 共识机制管理

2.2.1 PoSv3 共识机制

关于共识机制孰优孰劣的讨论一直在进行中，最常被讨论的共识机制有：工作量证明机制 PoW 、权益共识机制 PoS 、动态权益共识机制 和 HyperLedger 的拜占庭容错机制。共识机制的目的是利用分布式算法达成数据的一致性。正如 Fischer Lynch & Paterson 定理指出，除非所有节点达成 100% 一致，否则无法达成共识。在比特币网络中，矿工通过工作量证

明机制哈希碰撞参与交易验证。当矿工计算出满足一定条件的哈希值时，矿工可以向全网宣布新区块的诞生：

M

Hash(BlockHeader) ≤ D

M 是矿工数量，D 是挖矿难度，Hash() 代表 SHA256 哈希运算，输出值范围为 [0, M]。比特币的 SHA-256 哈希运算满足快速验证（方便网络中每个节点对运算结果进行验证）、可调整的难度值（可以根据全网算力进行调整）和相对公平性（也就是说，每个矿工解决哈希计算的概率与其算力成正比）。

不同的随机数值对应长度为 80 字节的不同区块头信息。根据全网算力的变化，挖矿难度会进行动态调整。当多于一位矿工同时解决了同一哈希难题时，就会出现分叉。这时区块链全网需要决定哪个链为主链。在比特币系统中若出现分叉，拥有更多算力的区块链将被选为有效链。基于权益共识机制的区块链平台大多都是点点币 PeerCoin 演化而来，而点点币是基于早期的比特币核心代码开发。工作量证明共识机制有多种算法，如 Scrypt 、X11、Groestl 、Equihash 等。开发新的算法主要是为了防止单个实体发起算力累计攻击，并确保特殊应用集成电路 ASIC 无法大规模实施。Core 芯算链在最新的比特币源代码技术上使用了权益共识机制。

在传统的 PoS 交易中，新区块的产生必须满足下列条件：

Proof Hash < coins × age × target

在 ProofHash 中，StakeModifier 利用未花费输出和当前时间进行运算。通过这种方法， 攻击者可以利用可以通过累计大量币龄并发起双花攻击。币龄引起的另一个问题是节点在获得奖励后间歇性上线，而不是持续在线。因此在后续版本的 PoS 中，移除了币龄从而鼓励更多的节点同时在线。

由于币龄攻击和其他类型的攻击，原始的 PoS 实施面临很多的安全漏洞。因此在 Core 芯算链核心代码中使用了 PoS 3.0。理论上 PoS 3.0 对长时间节点在线进行奖励，并取消对持币者离线钱包的激励措施。

2.2.2 互惠权益证明 MPoS(Mutualized Proof Of Stake)

由于 Core 专注于智能合约和区块链计算，交易处理需要花更多的时间去和智能合约交互和执行，因此需要更动态的费用机制。这其中有影响 Core 的几个安全隐患。一个比较大的隐患是，攻击者可以通过支付昂贵费用执行恶意程序，但由于这些费用会归于区块生产者， 最终攻击者付出的费用会变得很小。基于现有 PoS 系统的加密货币由于不支持图灵完备的智能合约 VM，不会受到这类攻击的影响。Core 是第一个基于 PoS 和以太坊智合约的区块链，实现了规范在区块链上允许的计算用途和能力的 gas 机制。

Core 进一步对 PoSv3 做了一个修改，用于部分解决通过垃圾交易攻击而产生的无用 stake 问题。采用 PoSv3 一致的共识机制，但改变了区块奖励和交易费用的支付体系。与区块生产者会立即收到区块奖励和交易费用相比，新的方法是奖励和交易费用在网络中多个挖矿者之间互相共享。具体说来，区块生产者在生成区块的时候会收到总费用的 1/10，而经过预定的区块确认时间之后，可以接收到另外 9 个块的费用的 1/10。这个新的方法看起来改动很微小，但它解决了影响 Core 的几个安全隐患。一个比较大的隐患是，攻击者可以通过买入足够的币从而成为有可能的区块生产者，通过某些 EVM 操作码使网络变得容易遭受 DoS 攻击。这只需要很小的代价，甚至恶意交易需要消耗交易费用和 gas，攻击者可以通过支付昂贵费用执行恶意程序，但由于这些费用会归于区块生产者， 最终攻击者付出的费用会变

得很小。使用 MPoS, 就不会出现这种情况，产生区块是只能收到 1/10 的 gas 费用，除非他能再挖出连续的 9 个区块，否则其余 9/10 的 gas 费用会因分给网络上其他的挖矿者而丢失。因此会导致这种类型的垃圾交易攻击会变得很昂贵。

当一个区块产生时，区块生产者的奖励交易必须包含至少 10 个输出。第一个输出属于本区块生产者，可以接收自己用于 stake 的币和 1/10 的交易费用，其他九个输出属于 500 个区块之前的区块生产者。当一个区块产生时，增加一个奖励费用的接收者，同时减去另外一个接收者，因此总接收者数量一直有 10 个。区块生产者另外 9/10 的奖励将在 500 个区块之后，即 (+501, +502, . . . , +509) 每个区块奖励的 1/10。该技术也有一些不足，但结合其他的技术可以解决潜在问题。一个恶意的区块生产者可以选择接受他自己的很低 gas 价格的交易，从而最小化用于执行和攻击所花费的币。这可以通过动态调整最小的 gas 价格来避免，最小 gas 价格通过去中心化的自治协议 DGP 来实现。这种机制可以控制各种垃圾交易和合约，不管它们是不是恶意的。

2.3 抽象账户合约层和以太坊虚拟机的集成

芯算链的设计是基于比特币 UTXO 模型，一般智能合约平台是采用账户模型的，因为智能合约作为一个实体，需要一个标识，这个标识就是合约的地址，所以智能合约运行、管理都可以通过这个地址来操作。芯算链模型设计中加入了账户抽象层（Account Abstraction Layer, AAL），用于将 UTXO 模型转换成可供智能合约执行的账户模型，对于智能合约的开发者来说，虚拟机的账户模型相对简单。它支持合约余额的查询，还可为其他合约发送资金等操作。尽管这些操作看起来十分的简单和基本，Core 芯算链中的所有交易使用比特币脚本语言，想要在基于比特币 UTXO 模型的芯算链的账户抽象层中实现， 比想象中更复杂，更困难。AAL 在其基础上进行了拓展，加入了三个全新的操作符：

OP_CREATE 用于执行智能合约的创建，把通过交易传输的字节代码传递到虚拟机的合约存储数据库，并生成一个合约账号；
OP_CALL 用于传递调用智能合约所需要的相关数据和地址信息，并执行合约中的代码内容。该操作符还可为智能合约发送资金；
OP_SPEND 将当前合约的 ID 哈希值作为输入的交易 HASH，或发送到合约的 UTXO 的交易 HASH，然后使用 OP_SPEND 作为花费指令构建交易脚本。

2.3.1 EVM 集成

以太坊虚拟机（EVM）使用了 256 比特长度的机器码，是一种基于堆栈的虚拟机，用于执行以太坊智能合约。由于 EVM 是针对以太坊体系设计的，因此使用了以太坊账户模型（Account Model）进行价值传输。而芯算链的设计是基于比特币 UTXO 模型，所以芯算链使用账户抽象层（Account Abstraction Layer），将 UTXO 模型转换成可供 EVM 执行的账户模型。此外，Core 还加入了基于账户的接口，使 EVM 可以直接读取芯算链上的信息。需要说明的是，账户抽象层可以隐藏某些特定功能部署细节，并为增强互操作性和平台独立性建立关注划分。

在比特币系统中，只有当解锁脚本（ScriptSig）与锁定脚本（ScriptPubKey）验证通过后才能花费相对应的交易输出。举例来说，锁定脚本通常会把一个交易输出锁定到一个比特币地址上（公钥的哈希值），只有当解锁脚本和锁定脚本的设定条件相符时，执行组合脚本才会显示结果为真（系统返回值为 1），这样相对应的交易输出才会被花费。而在 Core 系统中，

我们更强调智能合约执行的及时性。因此我们在锁定脚本中加入了 OP_CREATE 和 OP_CALL 操作符。当 Core 系统检测到该操作符时，全网节点就会执行该交易。这样一来，比特币脚本扮演的角色，更多的是将相关数据传送至 EVM，而不仅仅作为一种编码语言。与以太坊执行智能合约一样，由 OP_CREATE 和 OP_CALL 操作符触发的合约，EVM 会在各自的状态数据库中更改其状态。

考虑到芯算链智能合约的易用性，需要对触发智能合约的数据以及数据来源的公钥哈希值进行验证。为了防止芯算链上 UTXO 所占比例过大，开发团队将 OP_CREATE 和 OP_CALL 的交易输出也设计成可花费的，OP_CALL 的输出可以为其他合约或公钥哈希地址发送资金。
首先，对于在 Core 上创建的智能合约，系统会生成一个交易哈希值用于合约的调用。新合约的初始余额为 0（目前不支持非 0 初始余额的合约）。为了满足合约发送资金的需求，Core 使用 OP_CALL 操作符来创建交易输出。合约发送资金的输出脚本类似于：
1: the version of the VM
10000: gas limit for the transaction 100: gas price in Core satoshis
0xF012: data to send to the contract (usually using the solidity ABI) 0x1452b22265803b201ac1f8bb25840cb70afe3303:
ripemd-160 hash of the contract txid OP_CALL
这个脚本并不是特别复杂，并且 OP_CALL 完成了大部分所需的工作。Core 定义交易的具体花费（不考虑 out-of-gas 的情况）为 OutputValue，即 GasLimit。具体的 Gas 机制会在后续章节中谈及。当上述输出脚本添加到区块链上时，该输出就与合约的账户建立了对应关系，并且体现在合约的余额中。余额可以理解为可供合约花费的总和。
此外，还可以通过 P2SH 和非标准交易（non-standard transactions）进行交易。当前合约需要与另外一个合约或公钥哈希地址进行交易时，该合约账户中可供使用的输出将会被消耗。这部分被消耗的输出在 Core 网络中用作交易验证，必须存在，我们称之为合约预期交易（Expected Contract Transactions）。由于合约预期交易是在矿工验证和执行交易时产生的，而不是由交易用户产生，因此不会通过全网进行广播。

合约预期交易的主要工作原理是通过 OP_SPEND 代码实现的。OP_CREATE 和 OP_CALL 有两种工作模式，当操作符作为输出脚本时，由 EVM 执行；当操作符作为输入脚本时，EVM 不会被执行（否则会导致重复执行），这种情况下 OP_CREATE 和 OP_CALL 可当作无指令操作。OP_CREATE 和 OP_CALL 接收由 OP_SPEND 传递的交易哈希值，并返回 1 或 0（即可花费或不可花费）。这里就体现出 OP_SPEND 在整个合约预期交易的重要性。具体解释，当 OP_SPEND 将交易哈希值传递给 OP_CREATE 和 OP_CALL，由 OP_CREATE 和 OP_CALL 比对该哈希值是否存在于合约预期交易列表中。若在，则返回 1，即可花费的；否则返回 0，即不可花费的。这个逻辑间接提供了完整安全的方式来保证合约的资金只能由该合约使用，与普通 UTXO 交易输出一致。
目前还有一个问题尚未解决：当合约中有多个可花费输出时，不同的节点可以选择不同的输出，这就导致了花费不同的 OP_CALL 交易。同时，为了避免 DoS（拒绝服务）攻击和尽可能的简化共识原则，Core 将标准 coin picking 算法进行了适当简化，命名为 Consensus-critical coin picking 算法。这样，全网节点只能选择合约中的同一个可花费输出，不同选择将会导致所在节点与 Core 主链的分叉，分叉上的区块将被认为无效区块。

综上所述，当 EVM 合约向公钥哈希地址或另一个合约发送资金时，将会建立一个新的交易。使用 Consensus-critical coin picking 算法，可以在合约可供使用输出池中选择最合适的交易输出。选中的交易输出将会作为执行单个 OP_SPEND 的输入脚本，而输出则是资金的目的地址，并将剩余资金发送回合约，同时更改可供消耗的输出。然后，这个交易的哈希值将会添加在合约预期交易列表中。当交易执行后，该交易会被立即添加到区块上。当链上的矿工验证并执行该交易后，合约预期交易列表将会被重新遍历，验证正确无误后，将此哈希值从表中删除。这样，使用 OP_SPEND 可以有效的防止使用硬编码哈希值来更改输出的可花费性。

Core 账户抽象层使 EVM 无需过度关注 coin-picking，只需要了解合约中的余额，并可以与其他合约甚至公钥哈希地址进行资金往来。这样，仅需对以太坊智能合约做稍许修改即可满足 Core 的合约运行需求。

2.3.2 新增标准交易类型

Core 芯算链中参考比特币脚本模板，新增了标准交易类型。在 Core 芯算链上部署新的智能合约的参考代码如下：

1：the version of the VM [Gas limit]
[Gas price]
[Contract EVM bytecode]
OP_CREATE
向链上合约发送资金的脚本为： 1：the version of the VM [Gas limit]
[Gas price]
[Data to send to the contract]
[ripemd-160 hash of contract transaction id]
OP_CALL
注：目前标准的交易类型并不能消耗上述交易输出。只有当交易输出哈希值在合约预期交易列表上时，才可被消耗，而且该消耗不会在 P2P 网络进行全网广播。

2.4 Gas 模型

当 Core 开发团队试图把图灵完备语言加入到比特币区块链上时，意识到并不能仅根据交易的大小来决定支付给矿工的费用。因为一个很小的交易也可能包含无穷循环，从而导致整个区块链系统的崩溃。因此，Core 借鉴了以太坊中 gas 的概念，在 EVM 上执行代码、进行交易都需要消耗一定数量的 gas。若执行结束后还有 gas 剩余，这些 gas 将被返还给发送用户。在代码的执行过程中，一旦 gas 被耗尽，将会触发 out-of-gas 异常，则当前调用帧所做的所有状态修改（包括永久内存，合约资金）都将被回滚，但是消耗的 gas 不会被回滚， 因为回滚操作也占用了全网的算力和矿工时间。

尽管 Core 借鉴了以太坊中的 gas 模型，但二者的 gas schedule（即执行 EVM 代码的 gas 花费）仍截然不同。这是由于有的操作在 Core 的 EVM 执行比以太坊要耗费更多的 gas，有的操作则相反。因此，Core 将根据现有的以太坊 gas 价格并参考执行代码时消耗的全网算力来制定 Core gas 价格方案。

在 Core 创建一个合约注资或者执行一笔交易，需要明确定义两个参数，即 GasLimit 和 GasPrice。GasLimit 定义了执行此合约可花费的 gas 总量；GasPrice 定义了以 Core 聪为计价单位的 gas 价格。这样，在 Core 上执行合约的总花费则可以通过 GasLimit 乘以 GasPrice 获得。如果总花费超过了该合约支付的费用，则被视作无效交易。合约资金中减去执行合约的总花费，剩余的部分称之为交易规模费用（Transaction Size Fee）。这和标准的比特币交易费用模型十分类似。为了确定两个交易的优先级，矿工需要关注两个变量：一个变量是交易规模费用占总交易费用的比例（通常以最小代币值/KB 计算）另一个变量是执行合约的 GasPrice。这样，基于权益证明机制，矿工可以任意选择执行重要性和回报高的交易。这样的费用模型使 Core 交易执行更像是现实中的交易市场，矿工和用户可以按照各自的需求选择不同交易确认时间和交易费用。

2.4.1 Gas 费用退还

在 UTXO 模型中，矿工收取的交易执行费用是无法灵活调整的。也就是说，当矿工发现执行交易比预期简单时，无法将部分费用退还给用户。因此，必须找到一种方式，能够退还部分费用，并且在出现 out-of-gas 的异常情况时，将交易回滚，并支付矿工相应的 gas 费用。Core 中利用在矿工的 coinbase 交易中添加新的交易输出来实现 gas 费用的退还。与此同时，Core 还加入了新的区块验证共识机制，用于保证这些退款交易记录在 coinbase 中。否则，矿工可以选择保留这部分费用。

交易发起方的 refund 脚本可以作为输入以供输出脚本参考。通过简单地复制输出脚本，即可完成费用退还。目前，出于安全考虑，只有标准的 pay-to-pubkeyhash 或 pay-to-scripthash 可以用于执行退款。后续开发团队会评估该做法的安全性，考虑将此限制解除。
作为参考，OP_CALL 的标准格式为：输入：（按照入栈顺序）

•花费交易哈希值（可选项）
•版本号（使用虚拟机的版本号，目前仅有版本 1）
•Gas Limit （执行合约可花费的 gas 总量）
•Gas Price （以 Core 聪为计价单位的 gas 价格）
•数据（用于合约执行的数据）
•智能合约地址

输出：（按照出栈顺序）

•可花费性（资金是否可以花费）

所以，CALL 脚本为：
1
10000
0xabcd1234 3d655b14393b66a4dec8ba043bb286afa96af485 OP_CALL
如果虚拟机执行结果出现了 out of gas 的异常情况，则可以使用 OP_SPEND 进行资金的赎回，而该输出可供区块上的下一个交易使用。需要说明的是，这个输出是由运行 vin[0].prevout 得到的通用地址交易脚本。在 Core 的早期版本中，只有通用地址交易（pubkeyhash Script）的发送人可以在虚拟机上进行资金往来。尽管 Core 虚拟机也可以执

行其他形式的
交易类型，但是 msg.sender 会在 EVM 中将被赋值为 0，而且任何 out-of-gas 和 gas 费用退还的的资金将会保留在当前合约中。

2.4.2 未消耗 gas 退还模型

从另一个角度来看，Core 的设计必须解决另外一个问题，即如何将未消耗的 gas 退还给合约。这样，用户可以放心的在合约中储存大量的资金以满足合约的正常运行需求，而没有使用的部分可以退还。

在区块上，使用 vin[0].prevous 执行交易的发起，并返回退还资金的地址信息。资金的退还机制在比特币交易费用模型基础上做了稍许调整：

gas_fee = gas_limit × gas_price tx_fee = vin − vout
tx_relay_fee = tx_fee − gas_fee refund = gas_fee − used_gas
需要说明的是，开发团队也曾考虑，使用“Credit_price”参数来衡量 tx_relay_fee 和 gas_price，以方便矿工确认交易的优先级。

当合约执行完毕时，gas token 的消耗就会从总资金中扣除（通过 gas_limit 和 gas_price 的乘积），剩余部分将会执行 gas return script 退还当前合约，并在 coinbase 交易中添加 vin[0].prevout 的通用地址交易输出信息（以便矿工获取他们的区块奖励）。只有通用地址交易的输出才可以得到 gas 的退还，否则退还的 gas 仍由矿工持有（在 out-of-gas 情况下，发送的资金仍将保留在合约中）。

目前在 Core 中仅支持单个 EVM 合约执行交易，所以不会出现两个合约执行同一个交易，并共享交易费用的情况。后续的开发过程中如果能够解决一些关键问题，我们可能会加入多个 EVM 执行同一个交易的功能。

2.4.3 重要的 Gas 临界情况

矿工一定要对合约 gas 资金的退还脚本多加留意。如果 gas refund 脚本的输出使区块大小超过最大值，则该合约交易无法被写入区块中，合约交易则由下一个区块重新执行。因此在执行合约前，矿工应该在候选区块（candidate block）中为 refund 脚本留下充足的空间。不遵循这条规则会导致合约多次执行 refund 脚本后才可发现无法在当前区块上执行该脚本。如果没有 gas 费用退还，则没有相对应的费用退还输出。
在交易费用中包含 gas_fee 是需要达成共识的。如果区块上添加的交易会导致 gas refund 出现负值，或者 gas_fee 小于交易费用，则该交易及所在区块视为无效。
如果在操作脚本中有多个 OP_CREATE 或 OP_CALL 操作符，则该交易输出视为无效。虽然这种设计方案限制了脚本的能力，但更容易解决递归和 EVM 多方执行的问题。这样， 可以通过静态分析来判断脚本的有效性，而不是通过执行脚本来判断。

2.5 分布式自治协议

分布式自治协议是通过智能合约实现的，可管理的每个区块链网络功能都由独立的 DGP 智能合约 (采用 DGP-template 描述) 控制，这意味着每个功能有独立的治理、授权机制以及内置限制条件。每个 DGP 都有一个非常简单的核心治理模式：即通过具有管理或治理席位

的地址来对提案进行管理，包括对投票席位本身和普通提案的管理。现在支持 0，1，2 三种类型的提案，分别对应：管理或治理席位增加、管理或治理席位删除、普通参数提修改。所有提案的设置只有具备管理席位的发送地址才有权限设置，具有管理或投票席位的发送地址才能进行投票。
对 2 类型提案，即普通类型提案的投票，至少支持以下项目:

1.每个 Core 虚拟机操作码对应的 Gas 价格
2.区块创建者可接受的交易对应最低 Gas 价格
3.区块大小
4.区块 gas 限制

具体的网络参数更改流程为：

1.设计 DGP 参数更改提案。提案中需要明确新的参数值、实施变更的起始区块以及投票过程所对应的区块数量；
2.向社区公布更改提案，并收集相关反馈；
3.根据社区反馈，对更改提案进行相应调整；
4.将最终版提案发送至 DGP 智能合约；
5.投票立即启动；
6.具有管理或治理席位的代理者可以发送一笔交易给 DGP 智能合约，对该提案进行投票；
7.在投票过程中，若提案未获得足够投票，则该提案被否决，不执行任何修改；
8.若提案得到足够同意票，则 DGP 分布式自治合约将提案中的相关数据存储在持久 RLP 特殊存储空间内。

提案会更新到相应的变量保存，设计的新参数会在 500 个块之后生效，以避免出现不必要的孤儿块或分叉，钱包和区块链上的所有节点可以定时检查 RLP 储存从而判断是否有新的更改发生。该操作并不需要通过 EVM 即可方便地访问 RLP 数据；
对于 0，1 类型的提案，即管理或治理席位增加和删除，其投票流程和普通提案类似，也是通过具有管理或治理席位的发送者投票来决定，不过当投票通过后席位的变化可以立即生效。

2.5.1 DGP 智能合约的主要变量

投票和提案合约使用变量 currentProposals 保存当前提案的投票统计，DGP-template 中提案的设置需要在变量 adminKeys 或 govKeys 中存在的密钥控制者才能设置。变量 paramsHistory 保存了所有通过提案的记录信息，保存的最近一个参数为当前区块链网络使用的参数。

2.5.2 DGP 智能合约管理和治理席位增减提案

分别对应类型 0，1 提案，通过 addAddressProposal 操作可以设置接受投票的提案，对于待投票提案的设置首先判断是否 govKeys，只允许具有管理席位的发送者才有权限设置一个待投票的提案。增加管理地址或治理地址提案的投票权只限于具有 adminKeys 和 govKeys 权限的发送者，当提案投票数大于在变量 activeVotesRequired 中 adminVotesForManagement 设置的值时，管理地址或治理地址会保存到该 DGP-template 的 adminKeys 或 govKeys 变量里，这样就相当于在区块链治理委员会中增加了管理或治理相对应的投票席位。

2.5.3 DGP 智能合约普通提案的投票和统计

对应类型 2 提案，只允许具有管理席位的发送者才有权限开启一个待投票的提案。待投票提案本身是一个智能合约，该合约的地址保存在将保存在变量 currentProposals 中。提案的投票通过 votes.push(msg.sender) 完成，这样可以记录所有投票及投票者的发送地址。当提案投票数大于在变量 adminVotesForParams 设置的值时，提案的投票通过，并清空当前投票设置。通过的具体提案保存在 paramsHistory。因此，投票成功后区块链核心程序可以通过 paramsHistory 来读取最新的参数，并在参数投票通过后的 500 个区块开始实施。

通过智能合约管理以上三种类型提案的投票，就实现了 DGP 协议对可管理参数的治理，由于是全网节点共识规则的一部分，只要提案生效，全网就会自动切换到新的规则执行。

3 结论

本白皮书介绍了 Core 芯算链提供的全新的区块链智能合约解决方案。我们具体介绍了 Core 芯算链如何进行交易处理并利用权益共识证明机制进行交易验证。此外，Core 芯算链账户抽象层连接 UTXO 流通层和智能合约平台层，通过 AAL 可以将 UTXO 模型转换成可供不同虚拟机执行的账户模型，比如以太坊 EVM，或是 x86 虚拟机。通过 AAL 首次实现了集成以太坊虚拟机和比特币未花费交易输出协议。

Core 芯算链使用基于权益共识机制节约了大量的算力成本，而以太坊仍然使用工作量证明机制。虽然以太坊也计划将共识机制更改为 PoS，但新版本何时推出也没有具体的时间表。Core 对 PoS 3.0 的奖励机制做了一个修改，用于部分解决通过垃圾交易攻击而产生的无用 stake 问题，这种新的 PoS 机制命名为 MPoS。另一方面，比特币未花费交易输出与以太坊账户模型相比拥有更好的拓展性。Core 芯算链结合了简单支付验证协议，推出了基于移动设备的智能合约解决方案，并通过去中心化和高度分布式 PoS 交易验证协议实现“面向移动端”策略。

Core 芯算链通过设计分布式自治协议 (Decentralized Governance Protocol，DGP) 为区块链网络提供了一种新的治理模式。通过内嵌到创世区块的智能合约来治理区块链网络的参数，并实现一个去中心化的网络自治机制，实现区块链网络的自动升级和快速迭代，而不用担心软硬分叉对网络和社区带来的影响。

为了使区块链的智能合约平台能力得到进一步的扩展，实现了 x86 高性能虚拟机，可以直接通过 C、C++、rust、go 等语言进行智能合约的编程，也可以支持其他可以在 x86 架构上运行的编程语言。虚拟机支持 x86 指令集，并简化了的状态存储和一致性校验方法，x86 标准库和可信库使得智能合约的编写和部署在灵活性和安全性上有所改善。

综上所述，智能合约是社会技术工具，因此必须将广泛的商业应用落地考虑在内。Core 芯算链的各类贴近现实生活的应用场景就是其坚实的佐证。Core 芯算链的“面向移动端”策略旨在通过支持高度分布式权益共识机制交易处理将现有的区块链技术推向全新的高度。与此同时，Core 芯算链还将充分考虑前端用户体验，并将应用层开发融入进智能合约生命周期管理中，而这在现有的区块链解决方案中并没有得到充分的关注。