《论文阅读报告》

区块链技术发展现状与展望

区块链特点：

去中心化，时序数据，集体维护，可编程（以太坊（Ethereum）平台即提供了图灵完备的脚本语言以供用户来构建任何可以精确定义的智能合约或交易类型），安全可信等。

一、 概述区块链与比特币的发展史及二者的关系

比特币是迄今为止最为成功的区块链应用场景。比特币本质上是由分布式网络系统生成的数字货币， 其发行过程不依赖特定的中心化机构， 而是依赖于分布式网络节点共同参与一种称为工作量证明 (PoW) 的共识过程以完成比特币交易的验证与记录。区块链技术为比特币系统解决了数字加密货币领域长期以来所必需面对的两个重要问题，即双重支付问题和拜占庭将军问题

二、 阐述区块链的基础 架构模型及其关键技术

区块链技术的基础架构模型：

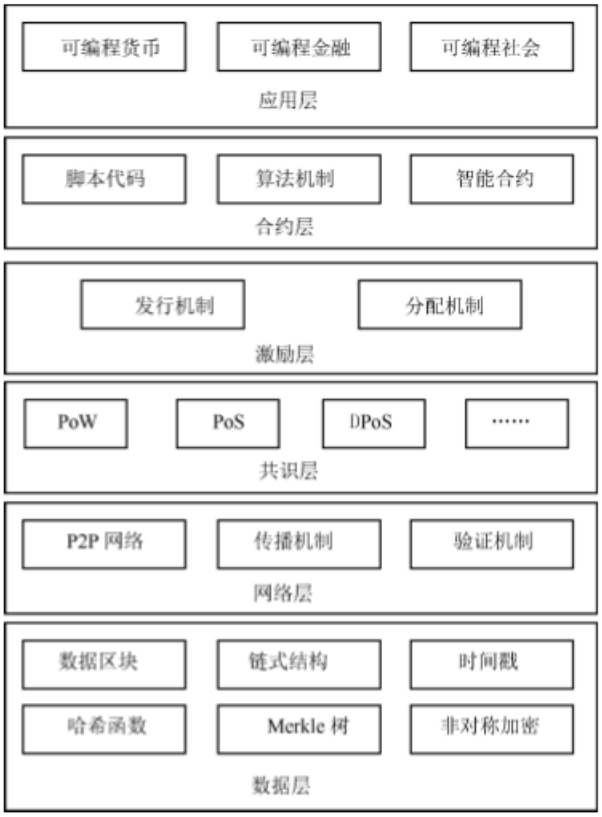

图 2 区块链基础架构模型

三、 概要总结了区块链技术的应用场景与现存的问题

区块链的应用场景：

区块链技术不仅可以成功应用于数字加密货币领域，同时在经济、金融和社会系统中也存在广泛的应用场景. 根据区块链技术应用的现状，本文将区块链目前的主要应用笼统地归纳为数字货币、数据存储、数据鉴证、金融交易、 资产管理和选举投票共六个场景。根据实际应用场景和需求，区块链技术已经演化出三种应用模式，即公共链、

联盟链和私有链.

区块链技术有待解决的问题：

- 安全问题
- 效率问题
- 资源问题
- 博弈问题

四、 介绍智能合约及其在区块链领域的应用现状

智能合约是区块链的核心构成要素(合约层)，是由事件驱动的、具有状态的、运行在可复制的共享区块链数据账本上的计算机程序，能够实现主动或被动的处理数据，接受、储存和发送价值，以及控制和管理各类链上智能资产等功能. 具体说来， 智能合约是一组情景—应对型的程序化规则和逻辑，是部署在区块链上的去中心化、可信共享的程序代码. 区块链和智能合约有极为广阔的应用场景. 例如， 互联网金融领域的股权众筹或 P2P 网络借贷等商业模式可以通过区块链和智能合约加以实现. 传统方式是通过股权众筹或 P2P 借贷的交易所或网络平台作为中心机构完成资金募集、管理和投资，实际操作过程中容易出现因中心机构的信用缺失而导致的资金风险.

五、 展望了区块链驱动的平行社会发展趋势

区块链为分布式社会系统和分布式人工智能 研究提供了一套行之有效的去中心化的数据结构、 交互机制和计算模式，并为实现平行社会奠定了坚实的数据基础和信用基础.

Ethereum_A_Secure_Decentralised_Generalised_Transaction_Ledger

一、 简介

首先以太坊最主要的目的是：以太坊的目的是基于脚本、竞争币和链上元协议（on-chain meta-protocol）概念进行整合和提高，使得开发者能够创建任意的基于共识的、可扩展的、标准化的、特性完备的、易于开发的和协同的应用。简而言之，以太坊想要创建一个新的去中心化计算机系统，每个人都可以参与进来，并使用它。 一个去中心化的任何人可参与的计算机系统比特币已经实现了，所以是新的一个系统，除此之外，比特币是一种使用有限脚本语言来构建去中心化支付系统的加密货币，但是以太坊不仅如此。以太坊不仅仅是一种加密货币，还是一种全球化去中心计算基础设施，能够执行智能合约与程序代码（使用多种编程语言编写）进而控制数字资产。在以太坊平台之上，开发者可以自行构建去中心化应用，这是它的不同之处。

二、区块链

以太坊在整体上可以看作一个基于交易的状态机：起始于一个创世块状态，然后随着交易的执行状态逐步改变一直到最终状态，这个最终状态是以太坊世界公认的最权威的一个版本。

换句话说，在两个不同账户之间发生的交易才让以太坊全球状态从一个状态转换成另一个状态。

三、一些形式约定

四、块、状态、交易

世界状态是在地址（160 位的标识符）和账户状态（序列化为 RLP 的数据结构）的映射。以太坊就像一个去中心化的计算机，世界状态则是这台电脑的硬盘。

以太坊中所有的账户信息都体现在世界状态之中，并由世界状态树保存。如果你想知道某一账户的余额，或者某智能合约当前的状态，就需要通过查询世界状态树来获取该账户的具体状态信息。

以太坊中有两种账户类型：外部所有账户（EOA）以及合约账户。我们用来互相收发以太币、部署智能合约的账户就是 EOA 账户，而部署智能合约时自动生成的账户则是合约账户。每一个智能合约都有其独一无二的以太坊账户。

账户状态反映了一个以太坊账户的各项信息。例如，它存储了当前账户以太币的余额信息、当前账户发送过的交易数量，每一个账户都有账户状态。

以太坊中包含两类交易（事务）：消息调用与合约创建。每笔交易的执行都将引起机器状态。区块是一个相关信息的集合。

交易推动当前状态到下一状态的转变

总体而言，以太坊有四种前缀树：

1. 世界状态树包括了从地址到账户状态之间的映射。世界状态树的根节点哈希值由区块保存（在 stateRoot 字段），它标示了区块创建时的当前状态。整个网络中只有一个世界状态树。
2. 账户存储树保存了与某一智能合约相关的数据信息。由账户状态保存账户存储树的根节点哈希值（在 storageRoot 字段）。每个账户都有一个账户存储树。
3. 交易树包含了一个区块中的所有交易信息。由区块头（在 transactionsRoot 区域）保存交易树的根节点哈希值。每个区块都有一棵交易树。
4. 交易收据树包含了一个区块中所有交易的收据信息。同样由区块头（在 receiptsRoot 区域）保存交易收据树的根节点哈希值；每个区块都有对应的交易收据树。

五、燃料和成本

由于以太坊是图灵完备的系统，为了避免计算资源被滥用，以太坊中所有编程计算操作都要收取交易费。计算就要投入成本，需要的计算资源越多则与之对应的交易费就越高。用于购买 gas 的以太币被转入受益者地址。如果（交易发送方）账户余额地址不能支付 gas 的费用，那么该交易就被认为是无效的。在以太坊平台中，只有执行交易的过程中才涉及 gas 消耗。

每笔交易都有一个与之关联的具体 gas 消耗量。gasLimit 与 gasPrice 也在交易中指定。

- gasLimit: 发送方愿意支付用于交易执行的 gas 最大数量。gasLimit 的存在,有助于解决交易陷入无限循环而无法退出的情况。在交易执行之后，如果仍有 gas 剩余，那么这些 gas 将返回给发送方。但是，如果交易因为某种原因执行失败，gas 就不再退回。
- gasPrice: gasPrice 是指 “你想支付多少以太币来购买一单位 gas”。交易发送方可以任意指定 gasPrice 的具体数值，然而，矿工也可以自由忽略一些 gasPrice 不符合他们需求的交易。

比特币交易费是由矿工收取的一小笔款项。比特币交易费并不是必须的，但由于矿工可以自由忽略任意交易，添加手续费则可以激励矿工将你的交易打包进区块链中，而且越高的 gasPrice 发出的交易更吸

引矿工，从而使得交易被更早打包。。比特币交易费的数值等于交易输入减去输出所得到的差值。

六、交易执行

交易执行是以太坊协议中最复杂的部分：定义了转换函数 Y。这个状态转移函数可以定义如下：

1. 检查交易的格式是否正确（即有正确数值）、签名是否有效和随机数是否与发送者账户的随机数匹配。如否，返回错误。
2. 计算交易费用:fee=STARTGAS * GASPRICE，并从签名中确定发送者的地址。从发送者的账户中减去交易费用和增加发送者的随机数。如果账户余额不足，返回错误。
3. 设定初值 GAS = STARTGAS，并根据交易中的字节数减去一定量的燃料值。
4. 从发送者的账户转移价值到接收者账户。如果接收账户还不存在，创建此账户。如果接收账户是一个合约，运行合约的代码，直到代码运行结束或者燃料用完。
5. 如果因为发送者账户没有足够的钱或者代码执行耗尽燃料导致价值转移失败，恢复原来的状态，但是还需要支付交易费用，交易费用加至矿工账户。
6. 否则，将所有剩余的燃料归还给发送者，消耗掉的燃料作为交易费用发送给矿工。

七、合约创建

当我们说一个交易是“合约创建”，是交易的目的是创建一个新的合约账户。

为了创建一个新的合约账户，我们使用一个特殊的公式来声明新账户的地址。然后我们使用下面的方法来初始化一个账户：

1. 设置 nonce 为 0
2. 如果发送者通过交易发送了一定量的 Ether 作为 value，那么设置账户的余额为 value
3. 将存储设置为 0
4. 设置合约的 codeHash 为一个空字符串的 Hash 值

一旦我们完成了账户的初始化，使用交易发送过来的 init code（查看”交易和信息”章节来复习一下 init code），实际上就创造了一个账户。init code 的执行过程是各种各样的。取决于合约的构造器，可能是更新账户的存储，也可能是创建另一个合约账户，或者发起另一个消息通信等等。

当初始化合约的代码被执行之后，会使用 gas。交易不允许使用的 gas 超过剩余 gas。如果它使用的 gas 超过剩余 gas，那么就会发生 gas 不足异(OOG)常并退出。如果一个交易由于 gas 不足异常而退出，那么状态会立刻恢复到交易前的一个点。发送者也不会获得在 gas 用完之前所花费的 gas。

不过，如果发送者随着交易发送了 Ether，即使合约创建失败 Ether 也会被退回来。

如果初始化代码成功的执行完成，最后的合约创建的花费会被支付。这些是存储成本，与创建的合约代码大小成正比（再一次，没有免费的午餐）。如果没有足够的剩余 gas 来支付最后的花费，那么交易就会再次宣布 gas 不足异常并中断退出。

如果所有的都正常进行没有任何异常出现，那么任何剩余的未使用 gas 都会被退回给原始的交易发送者，现在改变的状态才被允许永久保存。

八、消息调用

消息调用除了转变到新的状态和交易子状态，还用字节数组 o 表示输出数据，执行交易时输出数据是被忽略的，但在消息调用时，输出的数据可以由虚拟机代码执行时进行初始化。

由于没有新账户被创建，所以消息通信的执行不包含任何的 init code。不过，它可以包含输入数据，如果交易发送者提供了此数据的话。一旦执行，消息通信同样会有一个额外的组件来包含输出数据，如果后续执行需要此数据的话就组件就会被使用。

九、执行模型

协议实际操作交易处理的部分是以太坊自己的虚拟机，称之为以太坊虚拟机(EVM)。

像之前定义的那样，EVM 是图灵完备虚拟机器。EVM 存在而典型图灵完备机器不存在的唯一限制就是 EVM 本质上是被 gas 束缚。因此，可以完成的计算总量本质上是被提供的 gas 总量限制的。

此外，EVM 具有基于堆栈的架构。堆栈机器就是使用后进先出来保存临时值的计算机。
在执行特定的计算之前，处理器会确定下面所说的信息是有效和是否可获取：

- 系统状态
- 用于计算的剩余 gas
- 拥有执行代码的账户地址
- 原始触发此次执行的交易发送者的地址
- 触发代码执行的账户地址（可能与原始发送者不同）
- 触发此次执行的交易 gas 价格
- 此次执行的输入数据
- Value(单位为 Wei)作为当前执行的一部分传递给该账户
- 待执行的机器码
- 当前区块的区块头
- 当前消息通信或合约创建堆栈的深度

执行刚开始时，内存和堆栈都是空的，程序计数器为 0

然后 EVM 开始递归的执行交易，为每个循环计算系统状态和机器状态。系统状态也就是以太坊的全局状态(global state)。机器状态包含：

- 可获取的 gas
- 程序计数器
- 内存的内容
- 内存中字的活跃数
- 堆栈的内容

堆栈中的项从系列的最左边被删除或者添加。

每个循环，剩余的 gas 都会被减少相应的量，程序计数器也会增加。

在每个循环的结束，都有三种可能性：

- 机器到达异常状态（例如 gas 不足，无效指令，堆栈项不足，堆栈项会溢出 1024，无效的 JUMP/JUMPI 目的地等等）因此停止，并丢弃任何的更改。
- 进入后续处理下一个循环。
- 机器到达了受控停止（到达执行过程的终点）。

假设执行没有遇到异常状态，达到一个“可控的”或正常的停止，机器就会产生一个合成状态，执行之后的剩余 gas、产生的子状态、以及组合输出。