MultiVAC 编程模式紫皮书

支持图灵完备智能合约的区块链分片方案

MultiVAC 基金会
2019 年 4 月，版本 0.1

（本文由 MultiVAC 技术团队在应翔博士 shawn@mtv.ac 的指导下共同完成。
在此感谢李舸、董重、何亮、林楠、黄舒志、慎力博、马征远等对本文的贡献）

区块链的扩展性是阻碍区块链大规模应用的主要瓶颈，而分片是被认为最有可能打破这一瓶颈的技术之一。MultiVAC 继在黄皮书中推出了全球首个在计算、存储和传输三个维度都实现了完全分片的区块链架构，并依此实现了近乎完全的吞吐量线性扩展性后，对底层架构进行了进一步的拓展，使得 MultiVAC 在保证了高安全性，高扩展性，高效率和对矿工低门槛准入的同时进一步支持了图灵完备的智能合约，为大规模商业应用的底层平台支持构建了坚实的基础和保障。

MultiVAC 通过扩展数据的存储类型，并通过在协议层面进行约束的方式保证了跨分片交易的单调一致性，并允许开发者自由灵活选择符合业务逻辑的数据类型，提供了一致性的同时又尽可能降低了性能的损耗。同时又提供了一个新的用户分片方案，方便将吞吐量在所有分片上进行分流，并尽可能鼓励用户减少跨分片的交互。本文将介绍 MultiVAC 如何通过一系列精巧的技术设计，在保持了高效低门槛的全分片架构的基础下提供了对图灵完备的智能合约的支持。

1. 介绍

绝对的去中心化和低延迟几乎就是天平的两端。想象两种极端的情况：绝对的中心化，即拥有一台可以无限垂直扩容且永不宕机的超级计算机，所有的计算和存储都在这台计算机上解决，这样的计算是最高效的；反之，绝对的去中心化，每个参与的端只执行一个操作，除此之外所有的计算和数据存储都依赖于其他的端，指令和数据还都要通过匿名网络传输，这样的系统效率一定是非常低。

所有的区块链系统或分布式系统，都是试图在天平两端之间找一个平衡点。需要注意的是，去中心化的形式本身并不是目标，去中心化的意义是提供一套所有参与者都能够平等进行监督、具备天然公信力的运行机制，区块链社区应该是一个平等而透明的社区。

一个理想的区块链架构应该既能满足当前参与者对于平等透明的需求，又能提供足够的运行效率和表达能力，使得开发者能够在基础架构上创造丰富的应用。理想的区块链还应该提供适度的灵活性，因为不同的应用场景对于去中心化和运行效率的需求不同，理想的设计应该能够支持开发者根据自身需求在天平的两端间作出调整。

众所周知，完美的分布式系统是不存在的。例如著名的 CAP 不可能定理，就指出分布式系统很难同时达到一致性、可用性和分区容错性，系统功能要根据实际需要进行平衡和取舍。而区块链架构作为典型的分布式系统也是一样，天然带有分布式系统的缺陷和难题。MultiVAC 的目标是提出并搭建一个行之有效的系统，在坚决维护去中心化核心价值的前提下，尽可能地达到高吞吐量和低延迟的目标。对于开发者而言，MultiVAC 致力于打造一套灵活丰富的开发环境，以支持社区应用程序的丰富性。对于用户而言，MultiVAC 将尽量降低参与门槛，减少矿工负荷，让人人都能参与区块链系统，实现全民治理社区。

当前，区块链世界的公链开发者们共同面临着以下挑战：

1. **吞吐量限制。**绝大多数公链架构所涉及的共识验证方式限制了其运行效率，无法达到高吞吐量。最典型的就是 PoW 共识，每一笔交易都需要全网验证，全网整体的处理能力只相当于单节点的处理能力，不仅效率低下，而且参与者所做的大量重复的 Hash 计算工作并不创造任何实际价值，还造成能源浪费。

2. **存储和传输瓶颈。**目前可见的公链架构尚未突破全账本的技术设计，系统要求全网每个节点都需要保存全网全部的交易记录，随着区块链系统运行时间的增加，账本越来越大，矿工在网络中提供服务和首次进入网络所需要的数据也都随时间线性无限提高，这使得系统对矿工的存储要求越来越高，整个网络将不可避免地走向专业矿机中心化，这在限制了区块链系统性能扩展的同时，也剥夺了普通用户参与网络的权力，削弱了区块链系统的去中心化价值。

3. **开发僵硬局限多。**某些公链技术方案为了简化系统设计，舍弃了图灵完备性，导致了开发者能够实现的业务逻辑严重受限。譬如，一些开发框架强制开发者必须使用 MapReduce 编程模型。也有部分公链提出了自己的开发语言，但此类开发语言只是现代高级开发语言的子集，不仅有重新学习的成本，而且无法支持高级的开发抽象如面向对象等，仅仅增加了开发者的负担，无益于开发者和应用拓展。

4. **违背去中心化。**如前所述，区块链系统应该是一套完全平等透明可监督的系统。为了解决运行和存储瓶颈，一些区块链方案主动选择了牺牲去中心化程度，采用了超级节点方案，使得完全的平等互信不再受到保障。MultiVAC 认为，这是不可接受的。

很多分片项目基于区块链 2.0 时代的浅薄认知，将区块链分片的层次划分为网络分片、交易分片、状态分片，而忽视了**下一代区块链系统所面临的真正挑战来源于对传统冯诺依曼计算机体系的革命性创新和重定义**。若真正视区块链为下一代去中心化的世界计算机，区块链系统应重建新的体系结构以应对吞吐量限制，设计全新的存储和传输策略来重定义世界计算机的数据流，提供全新的编程模式来适配全新的体系结构，坚守去中心化的初心以保障这一世界计算机不受任何单一个体的控制且永不被关停。

MultiVAC 设计了全维度平行分片的架构，真正将区块链这个世界计算机从单核升级到可无限扩展的多核系统，解决了关键性的吞吐量瓶颈。开创性的存储与数据控制权分离策略，颠覆了冯诺依曼结构依赖的存储框架，使分布式数据流可交互、可验证。全新的异步分片开发模式和 LLVM 开发平台，允许开发者在多核分布式体系下灵活开发，实现

各种编程逻辑。而低矿工门槛保证了普通用户可以自由加入 MultiVAC 系统，整个系统的控制权又完整掌控在普通矿工手中，保证了 MultiVAC 成为真正去中心化的区块链架构。2008 年中本聪在比特币白皮书中高喊的“One-CPU-One-Vote”理念，被 ASIC 矿机和中心化矿池所无情摧毁，但终将在 MultiVAC 上再次重建，并更加强大。

MultiVAC 系统的特色如下：

- MultiVAC 是一个图灵完备的分片区块链方案。为开发者提供灵活的编程开发框架以搭建复杂的应用。
- MultiVAC 是首个性能线性扩展的分片技术方案。异步分片架构为构建高 TPS 的区块链网络提供了基础。
- 降低了参与者的设备门槛。MultiVAC 网络的参与者不需要性能强劲的设备，使得社区更加平民化、去中心化。
- MultiVAC 的拓展性十分优异。目前的 MultiVAC 方案灵活、简洁、优雅，在当前基础上可以较简单地加入新的扩展功能，如闪电网络或隐私计算等。

综上所述，MultiVAC 突破性地解决了当今区块链世界的多项关键性难题，为建立一个理想的区块链社区打下了坚实的基础。

2. MultiVAC 技术综述

当前区块链世界最明显需要解决的问题就是交易吞吐量过低。目前学术界和产业界提出的几种提高吞吐量的方案都不尽如人意：

- 超级节点：作为一种部分中心化的解决方案，违背了区块链去中心化的本质。
- 链外扩展：包括多链、侧链、闪电网络和状态通道，更像是补丁式的解决方案而非底层的本质升级，再加上会引入新的安全隐患，实际效果并不理想。
- 链上扩展：包括有向无环图（DAG）和分片。有向无环图在现实运行环境中的挑战主要是网络风暴。而分片方案面临的是由将全网节点进行分片带来的安全性降低风险和跨分片之间数据一致性问题。

MultiVAC 选择了在分布式系统中已经被广泛使用并被证明有效的分片方向来提高区块链的吞吐量。我们认为，分片的安全性退化和数据一致性难题虽然是很大的挑战，但是可以通过技术架构突破、工程创新和合理的取舍来解决。事实上，MultiVAC 技术团队已经设计并实现了一套切实可行的架构设计和工程方案，将在本文中详细论述。

2.1. 区块链分片的挑战及应对

分片方案的核心是将网络切分为多个子网络，通过并行执行区块链的共识过程提升吞吐量。其难点主要来自于两个方面，**数据可靠性**和**数据一致性**。

2.1.1. 数据可靠性层面的挑战及应对

区块链作为一个去中心化系统，其数据可靠性要求远高于传统分布式系统。因节点之间不具备完全信任关系，且部分节点存在作恶、伪造、串通等拜占庭行为，节点之间的数据信任关系非常薄弱。在很多分片项目中，**状态爆炸**是一个极具挑战性的难题。一方面，由于分片可能导致区块链安全性退化，被切分后的区块链系统的整体安全性将由单分片的安全性决定。另一方面，由于系统性能提升导致产生的状态信息大规模膨胀，状态爆炸带来更严峻的数据完整性挑战。如何建立节点之间的数据信任关系在庞大的状态面前显得尤为重要。

有一些分片项目通过将出块节点的选择随机化，实现了出块权利在节点之间随机快速切换，大幅度提高了安全性。然而，计算合约所需要的数据却很难像出块权利一样在节点间快速转移，尤其是在出块节点难以被预测的情况下。简单的解决方案是所有节点都保存所有数据，这将给节点带来巨大的存储压力。而且即使我们假设所有节点都拥有足够强的存储能力，为了能及时记录新确认的交易，每个区块都需要在全网所有节点进行广播和同步，这又将造成巨大的网络传输负荷。一个区块链对矿工的存储与网络性能要求过高必然导致变相的中心化，这违背了区块链的初衷。

MultiVAC 的分片方案不同于现在几乎所有的分片方案，实现了跨越性的突破和创新：

1. **数据控制权和存储职责的分离。**MultiVAC 将分片内节点分为矿工节点和存储节点。存储节点用于存储具体的交易数据与状态，矿工节点仅需保存当前交易数据的一个摘要信息(Merkle Root)。这些有限的数据保证矿工节点可以进行数据的验证和交易的确认以及摘要数据的更新，并通过 VRF 随机函数动态切换分片保证出块节点的不可预测性，从而满足了区块链对安全性的需求。

2. **基于可信证明的数据交互机制。**MultiVAC 设计了一个精妙的基于 UTXO 账本结构和 Merkle Root 的分布式数据结构和存储传输方案，使得矿工节点仅需保留区块摘要信息就可以完成跨分片数据的提交，使用极精简的数据就能对大数据集进行高效和安全地校验、增量修改和跨片通讯。

3. **计算、存储和传输的全维度分片。**MultiVAC 在将计算分片的同时，也让单个分片的存储节点只需存储该分片的具体交易数据，且区块的具体内容仅在分片内进行广播。从而大幅降低了区块链系统和矿工的账本规模和网络传输量，使得区块链的吞吐量能够随着分片数量的增加而近乎于线性的增长，满足了区块链的可扩展性。全维度分片降低了区块链参与的门槛，使得人人参与成为可能，回归了区块链社区去中心化的本质。在耗能低、不牺牲安全性和保证去中心化的情况下，完成高效、可信、无限扩展的分片。

2.1.2. 数据一致性层面的挑战及策略

分片数据一致性层面的挑战在于，如何在分片架构下支持一个图灵完备的虚拟机。难点主要体现在：

- 保证跨分片数据修改的一致性
- 避免依赖全局数据可能造成的性能退化

数据修改一致性

现有方案基本都是对跨分片数据修改的一致性进行了一定的取舍：

- 有的项目选择了较强的一致性。通常选择在交易时进行跨分片同步，即跨分片发送交易时等待回执，并对涉及的数据加锁，直到收到回执后完成交易。此类方案虽然提供了较强的一致性，但十分依赖算法来确保加锁的粒度尽量小，同时还有回执丢失的风险。即使这两个问题都能得到完美解决，系统也由于交易阻塞和数据加锁而性能急遽下降。

- 有的项目选择了较弱的一致性。跨分片交易被拆分为多个子操作。比如，一笔从分片 A 到分片 B 的转账交易，被拆分为在分片 A 对转出账户进行扣款，和在分片 B 对转入账户进行汇款。扣款在交易被矿工第一次处理时即被执行，而汇款往往会被作为一笔交易发送到分片 B，由分片 B 的矿工负责执行。此类方案的第一个缺陷是汇款可能在网络传输中丢失，第二个缺陷是汇款交易的执行顺序完全由分片 B 的矿工决定，导致汇款丧失了单调一致性。第一个缺陷可通过工程上重复发送汇款交易来进行弥补，但除了 MultiVAC 外大多数分片项目都缺乏协议层面的可靠性保证。第二个缺陷则要求汇款类操作必须满足可交换性，即改变操作的执行顺序要得到同样执行的结果。诚然，在上述例子中，汇款是满足交换性的，但是在其他逻辑的智能合约中的操作却未必满足交换性。所以，有些项目通过对虚拟机指令集进行约束来确保此类操作的可交换性。这类项目放弃了智能合约的图灵完备性，使得可开发的应用逻辑受到了极大的限制。

如上所述，目前绝大多数方案都存在明显的局限。MultiVAC 力图克服这一技术难题，提出一套理想的方案，既保障高性能，也提供一套图灵完备的开发框架，甚至允许开发者根据业务逻辑需要进行弹性调整。该方案包含以下三个主要方面：

1. **MultiVAC 虚拟机支持一套图灵完备的指令集。**
2. **异步分片编程模式。**通过分片之间数据不共享和交互不堵塞，避免了数据加锁带来的性能瓶颈，同时给开发者提供了灵活方便的编程模式。
3. **根据对单调写一致性的需求，将跨分片的归并操作划分为两类。**对于满足交换性的跨片操作，矿工可以无视内容，低成本地快速加入存储中；对于必须满足单调一致性的操作会需要较高的传输和计算成本，从协议层面强制要求矿工按照顺序执行。开发者可以根据业务逻辑来选择操作的类型。这使得系统在一致性、性能和成本方面达成了灵活与平衡。

对全局数据的依赖

智能合约对全局数据的依赖会限制系统性能。为了达到并行处理的性能，分片之间不可能进行完全数据共享，但是全局数据要求所有的分片都对此数据有感知。如何处理对全局数据的依赖是所有区块链系统都要面对的难题。

我们通过一个事例来说明全局数据的应用场景和 MultiVAC 的解决方案。假设有一个电商平台的智能合约。购买者调用该智能合约从售卖者处购买商品，该商品的存货是有限的。为了避免购买数超过商品的存货，开发者需要在智能合约中记录商品的剩余存量。商品存量就是全局数据，每次合约调用都会读写此数据。为了保证数据的一致性，通常有两种方案：

1. 给该数据加锁。数据同时只能被一个分片读写。
2. 数据只允许被单一指定分片调用。

显然，这两种方案都会导致智能合约的吞吐量制约于单个分片吞吐量。针对这一个难题，MultiVAC 提出了解决方案：

1. 将全局数据拆分到独立的分片上，从而将智能合约对全局数据的依赖转移到分片数据上。比如在电商智能合约中，可以将存量分配到多个分片上，并使得智能合约读写对应分片的数据，从而保证了在不超额售卖的前提下，充分发挥分片的并行性能。
2. 针对全局数据在各分片分配不均的问题，MultiVAC 允许用户在所有分片都进行操作。比如在电商智能合约中，全局额度被分配给各个分片，但由于不同分片的购买力不同，部分分片的额度会比其他分片的额度更快地耗尽。MultiVAC 通过允许用户在所有分片都进行购买，将全局交易分散到多个分片上。当某个分片的额度耗尽时，用户可以选择在其他分片运行该智能合约，避免合约执行在不同分片上的不平衡。

MultiVAC 独创性的**全局数据分散式设计**，旨在充分发挥区块链分片的最大性能。各分片尽可能独立地运行，这是区块链系统性能突破的关键，为 MultiVAC 整体系统性能线性扩展打下了坚实的基础。

2.2. MultiVAC 的总架构图

综合上文介绍的技术特点，MultiVAC 可以被分为

- **用户模块。**用户模块由用户节点的构成，主要与用户进行交互，并向分片模块发送交易，是智能合约的使用者。
- **分片模块。**分片是组成 MultiVAC 底层架构的重要单元。每个分片会维护一条独立的区块链。每个分片都有一组独立完整的存储、共识和执行模块，并依靠这些模块来产生、确认和添加新的区块。分片模块还会将在本分片产生但属于其他分片的数据异步传给对应分片。分片模块内包含两种节点：存储节点和矿工节点。存储节点组成了存储模块，矿工节点组成了执行模块和共识模块。

– 存储模块。由存储节点组成，为用户模块的交易提供可验证的数据并将数据提供给矿工模块。

– 共识模块。运行一个高效的拜占庭容错算法，通过协同投票来确认当前分片内产生的新区块。

– 执行模块。包含图灵完备的虚拟机，负责根据数据和指令执行交易，并将多笔已经执行的交易写入新区块。

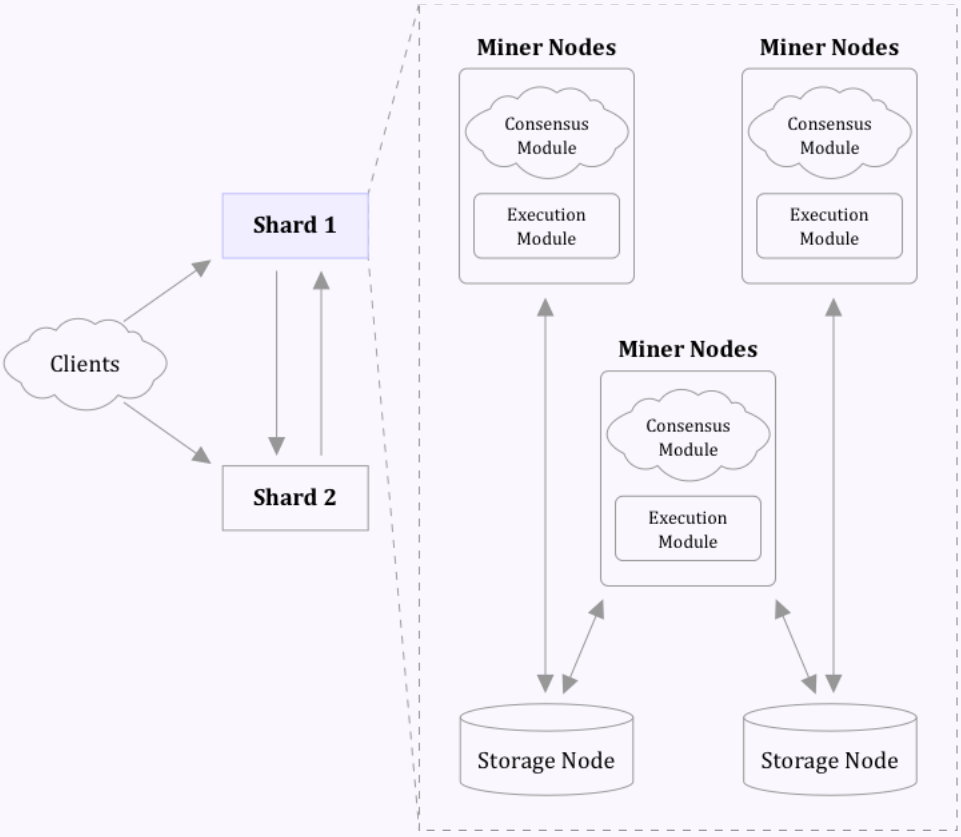

图1：MultiVAC 总架构图

MultiVAC 精巧地将各个功能模块做了合理划分，使运行节点各司其职。在存储上，将压力转移到存储模块，减轻了普通参与者的负担，而存储模块本身不涉及执行和共识，不存在中心化风险。在执行上，MultiVAC 提供了完整的指令集，不给开发者设置局限，且在设计过程中，时刻体现了对运行性能的重视。在共识方面，权力在各执行节点中均匀分布，平权原则落到实处。

更加重要的是，MultiVAC 团队认为，没有完美的区块链系统，一套好的区块链方案应该能够随着各领域的技术突破不断进化。所以在设计过程中，**MultiVAC 遵循了松弛可扩展的原则，保证整套技术框架易迭代易拓展，各模块能够灵活地整合新技术，与时俱进**。

接下来，本文将对在这一章中介绍的技术方案进行具体的解释。其中共识层面的方案已经在黄皮书中进行了详尽的介绍，包括：

- 节点根据不同的职能被分为用户节点，矿工节点和存储节点。
- 存储节点维护了一个保存了全状态的数据库。矿工节点仅仅持有该数据库的一个极为精简的摘要信息，并可依靠该摘要信息验证存储节点所提供数据的真伪。
- 矿工节点从存储节点处获取待确认交易以及其对应的信息，运行交易并产生一组对数据库进行修改的操作。并依靠算法在无需数据库具体信息的情况下，计算出更新数据库后新的摘要信息的值。
- 矿工节点在产生新的区块后，将该区块内产生的新数据的摘要以及属于其他分片的新数据拆分后发送给对应分片。对应分片可以在只收到摘要的情况下验证新数据的真伪。

本文假设读者对以上黄皮书中介绍过的技术内容有一定了解。限于篇幅，在本文中不会详细介绍上述技术细节，而将重点放在介绍 MultiVAC 如何解决执行层面的挑战，并成功实现了第一个可线性扩展的图灵完备的区块链架构。

3. 智能合约的部署与调用

与传统的智能合约系统相比，MultiVAC 智能合约的部署与调用过程中需要考虑到分片的存在。用户在注册的时候都会获得一个公私匙对，用户可以用这个公私匙对在任意分片调用智能合约，从而尽可能将智能合约的调用平均分摊到所有的分片下，并充分利用分片带来的并行性能提高运行效率和吞吐量。

为此，一个智能合约的代码需要被部署在所有的分片上。用户通过给智能合约的地址发送交易来调用智能合约的函数，并在交易中申明其调用的分片。交易会被发送到对应分片的存储节点，并由存储节点提供数据后由矿工节点运行并共识，并最终完成计算以及对存储状态的改变。

读者可以想象一个超级计算机中心，内部有着十台超级计算机供给中心的研究者使用，每台机器每次只能有一位研究者可以使用。

1. 假设不同计算机上安装的软件不同，比如只有一台计算机安装了某种特定的软件，并且所有研究者都需要使用这个软件。那么他们会全部等待这台计算机，而其他的计算机将由于无人使用而造成资源浪费。
2. 假设研究者并不能自由选择自己使用的超级计算机，某台计算机只能由一个人使用。那么当那个研究者没有使用该计算机的时候，该计算机会处于空闲状态，从而造成资源浪费。

MultiVAC 要求智能合约部署在所有的分片上，即解决了第一个假设带来的资源浪费；**允许所有用户在任意分片调用智能合约**，即解决了第二个假设带来的资源浪费。这

个章节会着重介绍 MultiVAC 如何将用户和智能合约布置在所有的分片，并且因此尽可能地利用分片带来的并行性能。

3.1. 用户与智能合约的交互

用户节点通过给分片的存储节点发送 Transaction 来调用智能合约的功能。一条 Transaction 包含交易的发送者，交易的接受者（智能合约地址），调用交易的分片，调用的函数，以及函数所需的参数和所需要数据的索引。用户可以自由选择该交易在哪个分片上被执行。

Transaction 首先会由对应分片的存储节点接收，并由存储节点根据交易的具体内容来提供具体的 DataCell[1]以及它们的 Merkle Path 并提供给矿工节点。矿工会通过 Merkle Path 检查数据的真实性，然后在执行模块的虚拟机上运行，并将结果写入区块，最终在通过共识模块达成共识后把区块加入区块链。具体的数据验证和共识流程可以参考黄皮书中相应的章节，在虚拟机中运行交易的流程会在后文关于智能合约的执行模块中介绍。

3.2. 智能合约的部署

一个智能合约需要能够在多个分片上被调用，否则就会导致智能合约的吞吐量被单个分片的吞吐量限制，从而丢失了单个合约吞吐量的线性扩展性。所以我们需要能够将智能合约部署在所有的分片上，并且大部分对智能合约的操作都可以在任意分片上运行，从而最大限度地保证了分片带来的并行和线性扩展效果。

当一个用户希望部署一个智能合约时，他会在某个他所选定的分片 i 发起交易调用一个内置合约。当该交易被矿工确认后，矿工会为该交易创建一个地址并分配给该智能合约作为其永久地址，并根据部署时提供的参数对该智能合约进行初始化：

- 在所有的分片创建一个分片数据[2]用来保存智能合约的代码；
- 在所有的分片创建一个分片数据并根据智能合约的逻辑进行初始化；
- 在主分片 i 创建一个全局数据[3]并根据智能合约的逻辑进行初始化。

当该交易被矿工确认，并由对应分片的矿工将产生的分片数据和代码数据添加入存储节点之后，用户就可以开始在相应分片调用该智能合约了。矿工如何将数据添加入存储节点的细节请参考黄皮书关于共识和存储的内容。

由于调用全局数据的交易必须在主分片运行，为了避免某个分片被太多智能合约定为主分片，MultiVAC 也允许智能合约设定一个操作将全局数据转移到另一个分片 j，并将

---

[1] DataCell 为 MultiVAC 中用以存储智能合约数据的基本单位，DataCell 包含的具体内容在下一章“智能合约的数据模型”中有详细解释。

[2] 分片数据：特属于某个分片，作用域为所有属于该分片的用户。详见“智能合约的数据模型”一章。

[3] 全局数据：可能被全网任何一个用户读写的数据，作用域为全网用户。我们将拥有该合约的全局数据的分片称为该智能合约的主分片。详见“智能合约的数据模型”一章。

分片 j 设定为它新的主分片。从而尽可能将用户的流量平均导向不同的分片，避免某个分片过于拥堵。

3.3. 用户交易的分片

前文中提到，MultiVAC 允许用户可以在任意分片调用智能合约。这个设计的优点为：

1. 分流用户，避免用户集中在一个分片内交易；

2. 鼓励用户进行分片内的交易。

避免用户集中在一个分片内交易

如果每个用户被约束在某一个分片内调用智能合约，那么可能会出现一个智能合约在某个分片内有很多用户调用，而在另一个分片中调用数过少，从而导致智能合约的资源在不同分片中无法被平衡分配和合理使用。如前文所述的计算机中心的例子，如果研究者被限制自己所能使用的计算机，那么可能会出现某台计算机空闲，但大量等待中的研究者由于自己不被允许使用该计算机从而导致了资源浪费。

该问题可以通过允许用户在所有分片调用智能合约解决，从而尽可能将计算在不同分片中进行分流，避免某分片过于拥挤而导致的性能瓶颈。

鼓励用户进行分片内交易

考虑一笔普通的交易，该交易会有一个付款方分片，一个收款方分片。在完全随机的情况下，跨分片的交易应该是随着分片数的增加指数级接近 100%：

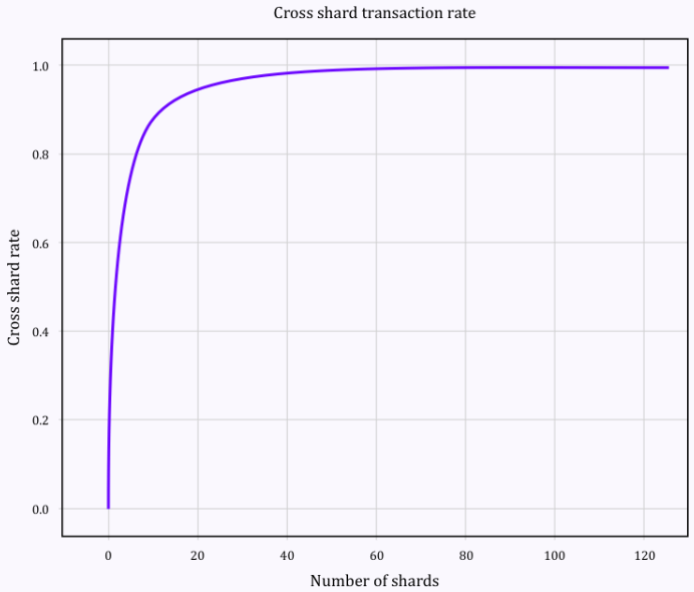

图2：跨分片交易所占比例与分片数量的关系

因为每次跨分片的交易都需要矿工在确认交易之后给收款方分片发送新产生的 DataCell，并且必须要产生的 DataCell 被合并入对应收款方所在分片的 Main Merkle Tree 之后才算交易彻底成功并可以被收款人使用，而分片内交易只需要被矿工确认就可以立刻被收款人使用了（关于具体的交易确认的部分请参考黄皮书）。所以跨分片的交易过多会导致交易的延迟增加，并且导致跨分片的数据量增加。

从用户的角度而言，跨分片交易除了带来更高的延迟，还会带来更高的使用费用（如 gas 费用），过多的跨分片交易对区块链和用户来说都是一个不必要的高成本行为。允许用户在所有分片都拥有一个账户可以有效地鼓励用户更多地进行分片内交易，不但可以降低分片之间的数据交互，还能够降低用户交易的延迟，提升用户体验。

4. 智能合约的数据模型

一个智能合约需要在区块链上保存一些状态。一个无状态的智能合约或者只能进行很局限的计算操作；或者依赖于用户提供数据，从而会引发信任危机和较大的网络流量负担。

在无分片的智能合约系统如经典以太坊中，状态的处理并不复杂，因为每一笔交易都会被所有的矿工执行，只要矿工对生成的块达成共识，交易运行完成后达到的状态就是一致的，无须额外的处理。然而 MultiVAC 通过将全网分割成若干个分片，把智能合约的计算，数据存储和网络交互分摊到多个分片中并行运行，从而提高了吞吐量，并且吞吐量可以随着分片数的增加得到近乎于线性提升的优点，但也使得**矿工不但要处理本分片的交易，还必须要根据在其他分片被运行的交易进行状态变化**，因此就需要引入相对复杂的状态同步机制。以下几个章节将对此详述。这一章首先介绍 MultiVAC 保存智能合约的状态数据的基本单元：**DataCell**。

为了便于读者理解，假设有一个智能合约的目的是创建一个部署在 MultiVAC 平台上的代币系统（以下称其为 XYZ 智能合约，它发布的代币称呼为 XYZ），并希望能够实现设定 XYZ 的总量，通过 MTV 兑换 XYZ，以及用户之间转移 XYZ 等代币系统必要的操作。在接下来的章节中，我们以这个代币智能合约为例，介绍 MultiVAC 的智能合约的分片架构。

MultiVAC 的状态数据会被保存在一种称为 DataCell 的数据结构中。每个 DataCell 都属于某个特定的智能合约，逻辑上来说所有属于某合约的 DataCell 的集合构成了该合约的总状态。 如在 XYZ 智能合约中，当前所剩余的 XYZ 额度以及每个用户持有的 XYZ 代币数量等信息就构成了合约的总状态。当智能合约被执行时会对 DataCell 进行添加删除或修改，并通过专门的同步机制保证数据的一致性。

在具体实现上，所有的 DataCell 数据会被组合为一个 Merkle Tree 保存在存储节点之内，矿工在使用数据的时候会向存储节点索取数据以及其对应的 Merkle Path。矿工会维护一个 Merkle Root 以便通过存储节点提供的 Merkle Path 验证数据的真实性。对于如何将数据保存为 Merkle Tree，矿工如何维护 Merkle Root，并且如何通过 Merkle Path 验证数据的技术细节请参阅黄皮书相应的章节。

在一个 DataCell 中会包含如下几个项，这些项的具体内容和目的将在这个小节接下来的内容中被一一介绍：

- DataCell 所属的智能合约地址
- DataCell 所属的用户地址
- DataCell 的类型：账户类型和 UTXO 类型
- DataCell 的作用域
- DataCell 的数据
- DataCell 所属的分片

4.1. DataCell 的智能合约地址与用户地址

每个 DataCell 内包含两个地址：智能合约地址和用户地址。这两个地址分别决定了该 DataCell 可以被哪个智能合约和哪个用户修改。

比如在 XYZ 智能合约中，一个用户通过兑换 MTV 获取了一部分 XYZ 代币，并产生了一个记录了该用户通过这笔兑换交易获取的 XYZ 代币的 DataCell。这个 DataCell 的用户地址将是该用户的账户地址，从而限制了这个 DataCell 只能被该用户花费；这个 DataCell 的智能合约地址将是 XYZ 智能合约的地址，从而避免了该 DataCell 被其他智能合约直接修改和使用。

值得一提的是，DataCell 的用户地址可以设置为空，从而允许任何用户对该数据进行修改。这种数据类似于一般程序语言中的全局数据，比如 XYZ 智能合约中用于存储 XYZ 代币在某分片中剩余额度的 DataCell，每个用户在用 MTV 兑换 XYZ 时都需要读取并修改该额度。后文会详细阐述 MultiVAC 如何保证该数据的一致性。

4.2. DataCell 的类型

在过去的区块链公链项目中，数据存储模型主要使用账户模型或者 UTXO 模型，两者分别有着不同的使用场景和优缺点。**MultiVAC 首度同时使用两种存储模型，将数据分成账户类型和 UTXO 类型两种，并允许开发者根据需要使用不同的类型来保存数据**。它们的性质和账户模型以及 UTXO 模型下的数据类似：

- 账户类型的 DataCell 唯一，不可被删除，在 Merkle Tree 中的位置固定，只能进行替换修改。
- UTXO 类型的 DataCell 可以有多份，每次使用后即被删除，位置不固定从而需要由使用的用户进行索引，不可被修改。

账户类型和 UTXO 类型的数据模型有各自的优缺点，分别适应不同的使用场景。

比如在进行跨分片交流时，例如 XYZ 智能合约有一个用户 A 从分片 1 对在分片 2 的用户 B 进行转账。如果 XYZ 代币数据是存储为账户类型的数据，那么转账交易需要被拆分为两步：从用户 A 的账户中扣款，将款项打入用户 B 的账户。网络的流量成本会翻倍，交易的延迟也会增加，确认交易的过程也会变得更为繁琐。而使用 UTXO 类型的数据，借助于黄皮书中介绍的矿工无须额外数据就可以更新其 Main Merkle Tree 的技术，可以大幅度降低跨分片转账的成本。

而在使用一些全局数据时，该数据更适合被保存为账户类型的数据。比如 XYZ 智能合约中在某分片 XYZ 代币剩余可兑换的额度。账户类型的数据自带的唯一，便于索引的优势使得它很适合用来存储全局数据。而如果使用 UTXO 类型的数据对全局数据进行存储，则会导致该数据极难被索引，以及其可能被创建多份或者被删除，从而带来严重的安全隐患。

4.3. DataCell 作用域

在一个分片架构下，和传统的分布式数据库一样，数据的一致性是一个很重要的考量。不同的数据往往会有不同的一致性需求。在 MultiVAC 中，数据被根据作用域分成了三种，并且保证了同作用域内的用户能够保持对该数据的读写一致性：

- **全局数据**：可能被全网任何一个用户读写的数据，作用域为全网用户；
- **分片数据**：特属于某个分片，作用域为所有属于该分片的用户。通常来说分片数据在所有分片都会存在一份，在不同分片读取的内容不一定一致；
- **个人数据**：只能被单一用户修改的数据。

对应到 XYZ 智能合约中：

- 全局数据：一个用来代表出售代币剩余额度的数据。
- 分片数据：每个分片含有一个用来代表每个分片上代币剩余额度的数据。
- 个人数据：每个用户拥有的代币。

显然的，全局数据和分片数据必须是账户类型的数据，而个人数据根据实际的需要可以是账户类型，也可以是 UTXO 类型。比如 XYZ 智能合约中个人拥有的代币更适合用 UTXO 类型的数据存储；而一些元数据，比如我们可以设定单个账户可直接兑换的最大额度限制，那么该额度限制就应该被存为一个账户类型的数据，因为这个额度限制应该是独立唯一的。另外，由于智能合约代码数据在分片内是唯一的（即同分片内不可能出现两份该数据），它也必须被设定为账户类型数据。账户类型数据的唯一性将由智能合约的代码进行保证。

值得一提的是，为了保证全网的读写一致性，全局数据将会只存在于一个分片内，对全局数据的调用必须发生在其所属的分片，在其他分片对全局数据进行调用会失败。过多的全局数据调用可能成为智能合约的性能瓶颈，所以开发者应该减少对全局数据的调用，可能将交易拆分到不同的分片。以 XYZ 智能合约为例，全局数据保存总额度，分片数据

保存某分片的额度，用户与分片数据交互来兑换货币使交易分散到各个分片。我们将拥有该合约的全局数据的分片称为该智能合约的主分片，其他分片称为该智能合约的子分片。

4.4. DataCell 的数据和所属分片

DataCell 的最终目的是为了存储一个状态数据。为了满足不同智能合约对数据的需求，MultiVAC 将数据保存为一个 DataCell 中的一个 JSON 格式的数据。开发者可以根据智能合约的需求在智能合约中保存不同的数据。

DataCell 中也会声明该 DataCell 所属的分片。这个数据只可能存在于其所属的分片中，被用户在该分片调用，并被该分片的存储节点保存。

5. 智能合约的跨分片交互

一个分片的区块链架构无法回避的难点是如何在处理跨分片的数据交互时，在数据一致性以及性能之间寻求到一个平衡。一个达成较强的一致性的设计往往伴随着复杂的交易流程和对数据的锁定，通常会大幅度拖累性能；而较低的数据一致性要求虽然能使得开发工作和区块链性能得到提升，但是往往会提高智能合约的开发难度，并带来潜在的信任和安全危机。

MultiVAC 提供了一个灵活自由且高效的跨分片交互方案。MultiVAC 提供了两种跨分片交互方式，分别为低成本的**跨分片增加 UTXO 类型数据**，和较高成本但保持了单调一致性的**跨分片修改账户类型数据**。开发者可以根据具体的商业业务需求从中进行选择，尽可能将高频的跨分片操作用较低成本的增加 UTXO 类型数据实现。

MultiVAC 将全网分割为多个分片模块，每个分片模块包含一组矿工和存储节点。分片模块的存储空间彼此独立，不进行任何的数据共享。分片之间进行交互的方式为彼此异步发送消息。每当一个分片完成出块以后，会根据确认的交易的计算结果生成一组对自己以及其他分片的存储节点进行数据操作的指令。矿工会直接执行自己分片内的数据操作指令，而将其余指令拆分后发送给对应的分片。对应分片产生新区块时由其所属的矿工进行对该分片内的数据修改。通过顺序处理来自其他分片的指令来保证跨分片数据的单调写一致性。

发送指令给其他分片的操作是一个异步操作，发送完毕后矿工可以立刻开始进入下一轮的出块，无需等待其他分片的回执。

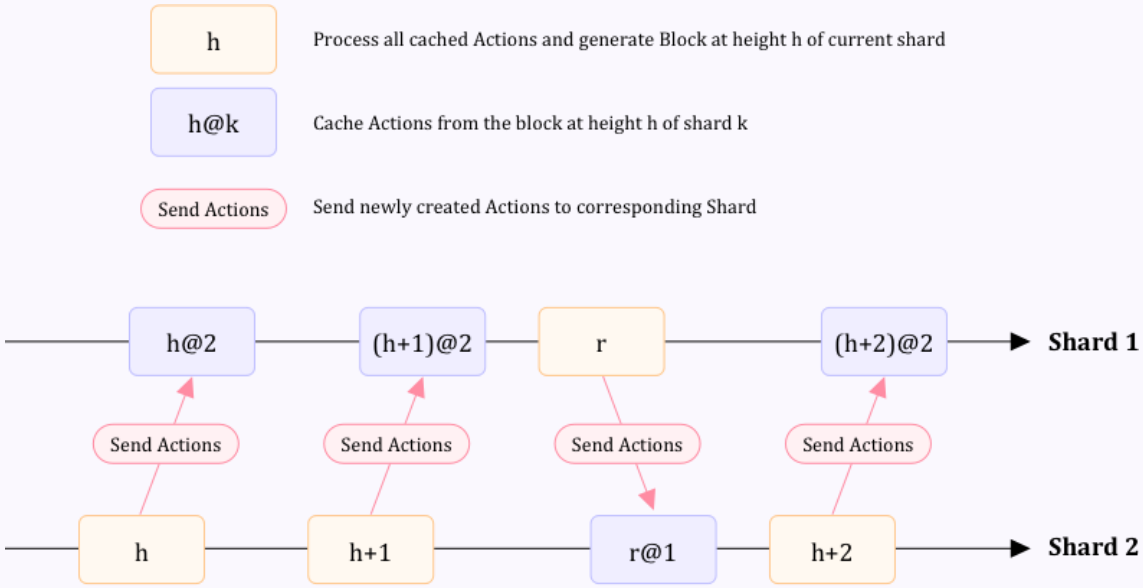

图3：分片异步发送区块确认消息流程图

本节剩余的内容将对跨分片的数据交互方案进行详细的介绍。

5.1. 传递数据修改指令的单元：Action

Action 是一个节点根据计算结果对存储节点发送的修改数据的请求。这个请求可以是分片内部传递的，也可以是跨分片传递的。Action 可以被分成四种：

- add：创建一个新的 DataCell。
- del：删除一个存在的 DataCell。跨分片无法进行 del 操作。
- update：将一个旧 DataCell 的数据更新为一个新的值，并保持数据的所属智能合约和用户不变。只有账户类型的数据才可以被它所属的分片的矿工进行 update 操作，跨分片无法进行 update 操作。换言之，update 操作在智能合约在本分片修改一个账户类型数据时被触发。
- merge：产生一个 DataCell 并发送到另一个分片，并在该分片内根据该 DataCell 运行一个指定的智能合约函数更新某个账户类型的数据。这个操作专用于跨分片的数据更新。merge 操作在跨分片修改一个账户类型的数据时被触发。

读者需要注意的是，Action 和 Transaction 是不同的操作。Transaction 是由用户端主动发起，发送给存储节点然后转发给矿工节点的消息，需要调用函数在虚拟机上运行，并需要通过共识，有可能会因为虚拟机指令码错误、Gas 用完等原因执行失败的指令；而 Action 是矿工根据交易的运行结果自动产生，并发往自己或其他分片的存储节点的指令，它是 MultiVAC 状态同步机制的一部分，只要 MultiVAC 区块链网络的基本假设仍然成立（例如作恶节点数不超过 1/3 等）以及智能合约中逻辑正确，Action 指令默认必然会被成功执行。

5.2. 跨分片操作的创建

在我们继续之前，请读者回顾一下黄皮书中关于矿工生成 Block Merkle Tree，并生成 Partial Merkle Tree 发送至各对应分片，以及对应分片的存储节点和矿工节点可以将新产生的 DataCell 添加入 Main Merkle Tree 的具体流程。由于篇幅问题，在这里我们省略具体细节。

MultiVAC 支持两种跨分片的操作：add 和 merge。

- add 主要用于在其他分片增加一个新的数据。比如在 XYZ 智能合约中，当用户进行跨分片的 XYZ 代币转账时，会先消耗该分片转账用户的某个 XYZ 代币数据，并生成一个新的，属于收款人的 XYZ 代币数据，并发往收款人所属分片。只要该数据被增加入收款人分片的 Main Merkle Tree 中，即代表交易完成。

- merge 主要用于更新其他分片的某个账户类型的数据。比如在 XYZ 智能合约中，当智能合约的管理者希望将全局数据中的额度分配给某个分片时，会发送一个交易给主分片。该交易首先降低全局数据存储的额度，并生成一个包含了转移额度的中转 DataCell，作为 merge 的参数发送给目标分片。由于某分片可供兑换的 XYZ 额度被存储在一个账户类型的分片数据中，该 merge 操作到达目标分片后，必须被目标分片的矿工通过执行一个归并的智能合约函数，利用该操作中附带的数据信息，增加对应的分片数据中的额度，才代表交易完成。

当矿工完成一个新区块以后，会生成一组 add 操作，以及一组 merge 操作。矿工会将它们根据区块，以及执行顺序排序，并生成两个 Block Merkle Tree。在这里我们分别称呼它们为 Add Block Merkle Tree 和 Merge Block Merkle Tree。矿工会将这两个 Merkle Tree 的根值记录入 Block Header 中并广播至全网。然后矿工会生成对应分片的 Partial Add Merkle Tree 与 Partial Merge Merkle Tree 并发给对应分片的存储节点，使得存储节点可以根据 Block Header 中的 root 验证并重建属于其分片的 Partial Merkle Tree。

值得一提的是，由于跨分片如何合并并修改数据的逻辑取决于具体的业务逻辑，所以 merge 指令会跨分片调用一个开发者自定义的智能合约函数。开发者在编写智能合约时需要负责保证当 merge 指令被创建时，其调用的智能合约函数必然会成功执行。

5.3. 跨分片操作的执行

当矿工节点收到来自其他分片的 BlockHeader 时，该节点会将区块头中的 Add Block Merkle Root 和 Merge Block Merkle Root 缓存，并在其所服务的分片产生下一个区块时将 Add Block Merkle Tree 添加入其所在分片的 Main Merkle Tree，并运行 Merge Block Merkle Tree 中的 Action。

使用 Add Block Merkle Root 在无具体树的细节的情况下添加 Add Block Merkle Tree 的方式详见黄皮书中的对应章节。

矿工在运行 Merge Block Merkle Tree 中的 Action 之前需要向存储节点索要具体的 Merge Block Merkle Tree 中包含的 Action 以及其所需要的数据（比如本分片需要被更新的

账本类型数据）。通过与索要缓存交易类似的方式，矿工可以从邻近矿工节点或者存储节点处获取完整的 Merge Partial Merkle Tree，并与从区块头中获取的 Merge Block Merkle Root 加以验证，确定 Action 的可靠性。

当矿工被选为本分片的出块者后，他会先将所有收到的所有未被加入本分片 Main Merkle Tree 的新区块头根据（分片，高度）进行全排序，然后根据顺序依次将对应分片对应高度的 Add Block Merkle Tree 加入 Main Merkle Tree，然后按顺序运行其所有的 Merge Block Merkle Tree 中的所有 Action。当将所有其缓存的新区块添加入 Main Merkle Tree 以后，矿工才会将本分片新产生的块添加入 Main Merkle Tree，并更新区块头部信息中的高度向量。

在跨分片交互方面 MultiVAC 拥有如下优点：

- **单调一致性**：通过由矿工将缓存的数据根据（分片，高度）进行排序，保证了高度较低的分片的 merge 操作必定在高度较高的分片之前进行；并通过对同区块内的 merge 操作进行排序，保证在区块内更早运行的交易所产生的 merge 操作必定在之后运行的操作所产生的 merge 之前被运行。从而 MultiVAC 保证了某分片对另一个分片进行的跨分片数据修改时的单调一致性。

- **添加和修改的分别处理**：矿工进行 add 操作时，仅需要一份区块头中的 Add Block Merkle Root 即可；而进行 merge 操作时，需要从临近节点或存储节点处索取具体的函数操作信息以及需要被归并的数据，会消耗更高的流量和计算资源。开发者可以通过取舍来取得更高的性能和更低的成本。比如在 XYZ 合约中，更高频但是对一致性相对要求较低的跨分片转账交易被设计为一个跨分片的 add 操作，而较低频但是对一致性要求较高的跨分片修改额度的交易就被设计为了一个跨分片的 merge 操作。

6. 存储模块和共识模块

这一节我们会介绍分片模块内部的存储模块和共识模块，以及它们的交互。

6.1. 存储模块

MultiVAC 的存储节点起到了存储模块的作用。它负责存储它所属的分片内的 DataCell 数据，并根据新的区块更新当前存储的数据。它负责收取用户节点的交易信息，提供交易所需的数据，并转发给矿工节点进行执行。存储节点提供的是数据存储的服务，它们没有修改数据的权限，并且它们提供的每一个数据都会附加上足够的信息来证明该数据是真实可信的。关于存储节点如何保存数据，如何接收新数据，存储节点是如何进行数据的分片的，以及存储节点的数据的可信度是如何保证的细节，请读者参阅黄皮书相关的章节。

6.2. 共识模块

智能合约的矿工节点负担着两个任务：

- 运行交易并根据运行结果对数据内容进行修改，并将交易打包成一个区块；

- 共识某个节点产生的区块，确认合法性，并通过共识算法决定是否将该区块写入链。

这两个行为是彼此独立的，前者由矿工节点的执行模块负责，而后者由矿工节点的共识模块负责。

MultiVAC 的共识模块运行了一个基于 PoS 的拜占庭共识族来进行共识，用户需要锁定一定的资金作为押金从而参与拜占庭共识，并在最后收集到足够的签名后广播区块共识达成的消息，并将区块具体内容拆分并发送至需要的分片。在这里将不对具体的押金，矿工选拔和共识过程等细节流程进行描述，因为这些内容与黄皮书中介绍的交易模型区别不大。相关内容请读者参阅黄皮书相应章节。

6.3. 存储模块和共识模块的交互

矿工在出块以后，给存储节点（包括同分片和其他分片）发送的消息可以被理解为上文介绍的 Action。在实现上可能是整个块，也可以是一个块中属于该分片的 DataCell，但是本质来说都是在发送 Action 让存储节点修改它当前保存的 Merkle tree。

于是，一个共识的整体流程概括为：

- 用户节点根据需求向存储节点发送交易请求。
- 存储节点根据交易请求搜索数据库，提供运行交易所需的数据，并将交易和数据发送给矿工节点。
- 矿工根据交易和数据进行计算和共识，并根据计算结果返回一组对数据更改的请求 Action。
- 将 Action 发送给存储节点，存储节点根据 Action 修改数据。

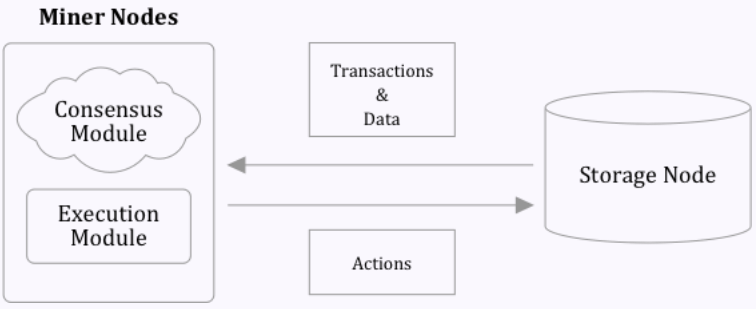

图4：矿工节点与存储节点的交互

在以上描述中，我们忽略了分片的存在。考虑分片后，数据的流动可以参考下图：

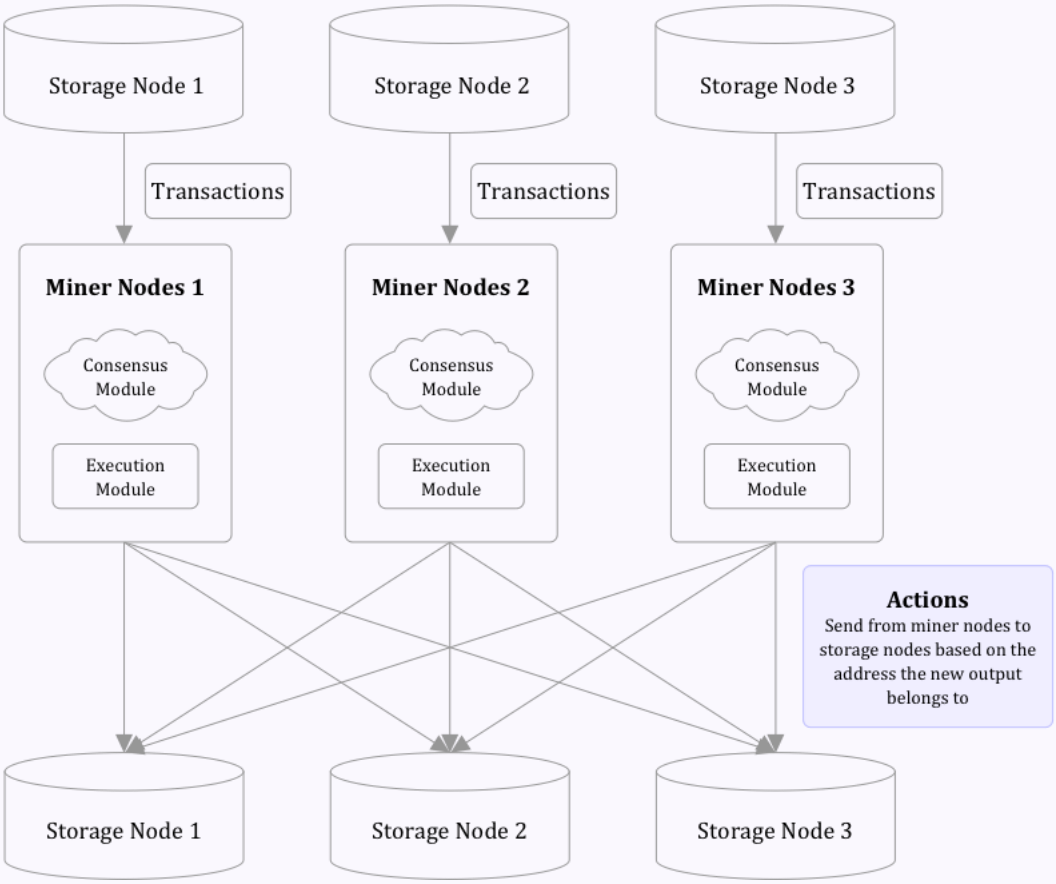

图5：跨分片矿工节点与存储节点的交互

7. 智能合约的执行模块

智能合约的执行模块是矿工节点用于处理智能合约调用的部分。它的核心是一台虚拟机，每个交易都会指定智能合约的地址以及调用参数。虚拟机加载智能合约的代码以及相关数据并执行被调用的智能合约的方法，产生一串对于数据的修改操作。最后执行模块将这些操作合并后记录在区块里，并交给共识模块进行共识。

智能合约的操作其实是对数据进行的增删改。一个智能合约不能没有任何限制地修改任意数据。MultiVAC 对智能合约的数据权限进行了一定的限制，从而尽可能保证了每个智能合约都只能直接修改自己的数据。对其他智能合约数据的修改必须通过调用对应智能合约公开的接口，保证如果有一个智能合约出现了错误，那么这个错误会被限制在该智能合约之内。

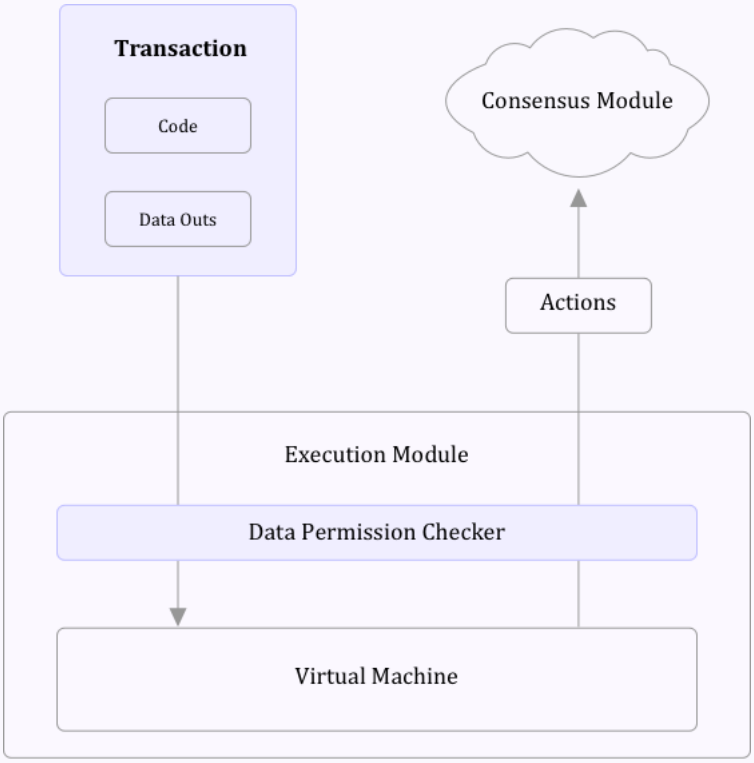

图6：执行模块与共识模块的交互

在 MultiVAC 中内置一个图灵完备的虚拟机。该虚拟机具有高效、快速、安全、开放、可调试的图灵完备的指令集。该指令集是 LLVM 的一个可移植编译目标。LLVM 本质是一个通用的编译器框架，可将 C/C++/Rust 等多种高级语言翻译为中间语言（IR），并使用 LLVM 成熟的优化模块进行优化，然后将优化后的 IR 翻译为目标硬件架构的指令。在该框架下，MultiVAC 的开发者可以根据其具体情况灵活选择多种高级语言，最大化的复用已有代码库。

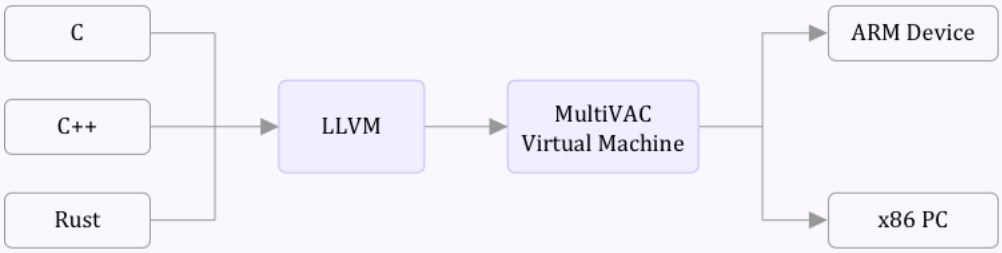

图7：LLVM 前端编译器与 MultiVAC 后端虚拟机

在执行器的选择上，目前 MultiVAC 选择基于软件的虚拟机来提供跨平台性从而使更多的用户可以参与进来。由于 LLVM 的架构允许切换目标架构，在将来 MultiVAC 还可推出针对硬件的智能合约执行模块，从而极大的提高智能合约的执行效率。LLVM 对多种语言提供的源码级别的调试以及性能剖析以帮助开发者快速地开发出高效代码。

在分布式系统中，为了避免恶意代码对网络平台的循环攻击，以及缺陷代码对计算资源的持续占用，执行智能合约的沙盒环境应该具备解决图灵停机问题的能力。在

MultiVAC 虚拟机内会采用类似于 Ethereum 中的 gas 计费机制来解决此问题，对合约需要执行的指令进行收费。当成功完成计算任务或者费用消耗完毕时，虚拟机会停止。

8. 结语

在这篇文章中，我们介绍了一个全维度分片，并支持了图灵完备性的智能合约的 MultiVAC 分片区块链架构。它可以在保持去中心化的前提下，最大限度地满足用户和开发者在速度和吞吐量上的需求，并可以随着用户的增加保持线性扩展性。作为下一代的智能合约分片区块链平台，MultiVAC 可以真正支撑起高吞吐量的商业化应用，将区块链技术真正推向实体经济。