Sui 智能合约平台

译者： moveworld社区— G_xbt

1,简介

Sui 是一个去中心化的无许可的智能合约平台，拥有倾向于资产的低延迟处理的特性。它使用 Move 编程语言将资产定义为对象(每个地址可以拥有这些对象)。Move 程序代码定义了对这些不同类型对象的操作，包括创建它们的自定义规则、将这些资产转移给新所有者以及改变资产状态的操作。

Sui 由一组无需许可的权限管理者(authorities)维护，这些权限管理者的作用类似于其他区块链系统中的验证者或矿工。它在权限管理者之间使用拜占庭一致的广播协议，以确保对资产的共同操作的安全性。同时，与单纯的拜占庭协议相比，拥有更低的延迟和更好的可扩展性。它只依赖拜占庭协议来保证共享对象的安全（译者注：Sui 的共识协议是混合协议，拥有两种(类似于poalkdot Gradpa和Babe 协议)，拜占庭协议仅仅是两种协议中的一种，这样做的好处是，可以结合两种的协议的优势）。同时，Sui 也拥有治理操作，检查点（译者注：Sui 和传统的区块链项目不同，没有区块这个概念，对数据的检索会比较复杂，所以设置这个检查点，用来方便数据的查询，每一个epoch 结束，就会设置一个检查点，每一个epoch的时间是24小时)，和在关键延迟路径上执行。在可能的情况下，部署在Sui上的智能合约可以并行执行。Sui 也支持可以验证交易数据读取的轻客户端以及可以检查所有状态转换的完整性的完整客户端。这些设施允许建立与其他区块链的信任最小化桥梁。

Sui的原生资产 SUI 用于支付所有操作的 gas。SUI的持有者可以使用SUI来将权益委托给权限管理者，让这些权限管理者拥有在epoch 时间内操作 Sui的权利，并且Sui会定期根据委托给权限管理者的权益的数量重新配置权限大小。

本白皮书被组织成两部分，第二个部分描述了使用 Move 语言和 Sect 的 Sui 编程模型。第四部分描述了确保 如何Sui 安全、活跃和性能的无许可去中心化系统的具体运行细节。

2，Sui 智能合约编程

Sui 智能合约是用 Move[4] 语言编写的。Move语言是安全且富有表现力的，其类型系统和数据模型自然支持使 Sui可扩展的并行协议/执行策略。Move语言是一种用于构建智能合约的开源编程语言，最初是在 Facebook 为Diem区块链开发的。该语言与平台无关，除了被Sui采用之外，它还在其他平台（例如 0L、StarCoin）上越来越受欢迎。

在本节中，我们将讨论Move语言的主要功能，并解释如何使用它在 Sui上创建和管理资产。关于Move功能的更详尽解释可以在 《the Move Programming Language book 》[1]中找到，更多特定于Sui的Move语言的内容可以在 《the Sui Developer Portal 》[2]中找到，在本白皮书中关于Sui更正式的Move语言的描述可以在第 3 节中找到。

2.1 概述

Sui 的全局状态包括一个由 Move包创建和管理的可编程对象池，这些包是包含 Move 函数和类型的 Move 模块（详见第 2.1.1 节）的集合。因此，Sui 对象可以分为两类：

• **数据结构值**：类型化数据由 Move 模块控制。每个对象都是一个结构值，其字段可以包含原始类型（例如整数、地址），其他对象和非对象结构。

• **软件包里代码的值**：一组作为原子单元发布的相关 Move 字节码模块。包中的每个模块既可以依赖于该包中的其他模块，也可以依赖于先前发布的包中的模块。

对象可以对资产进行编码（例如，可替代或不可替代的代币）、授予调用某些函数或创建其他对象的权限的能力、管理其他资产的“智能合约”等等——由程序员决定。声明自定义 Sui 对象类型的 Move 代码如下所示：

```
struct Obj has key {
  id: VersionedID, // globally unique ID and version
  f: u64 // objects can have primitive fields
  g: OtherObj // fields can also store other objects
}
```

所有表示 Sui 对象的结构体（但不是所有 Move 结构体值）都必须具有 id 字段。关键字“key”[3]指示该值可以存储在 Sui 的全局对象池中。

2.1.1 模块。一个 Move 程序被组织成一组模块，每个模块都包含一个结构声明和函数声明的列表。一个模块可以从其他模块导入结构类型并调用其他模块声明的函数。

在一个 Move 模块中声明的值可以流入另一个模块——例如，上面示例中的模块 OtherObj 可以在与定义 Obj 的模块不同的模块中定义。这与大多数智能合约语言不同，后者只允许非结构化字节跨合约边界流动。Move 能够支持这一点，因为它提供了封装特性来帮助程序员编写健壮安全的 [14] 代码。具体来说，Move 的类型系统确保像上述 Obj 这样的类型只能由声明该类型的模块内的函数创建、销毁、复制、读取和写入。这允许模块对其声明的类型的变量保证不会被改变，即使它们流过智能合约信任边界，这些变量也会继续保持。

2.1.2 交易和入口点。全局对象池可以通过发送交易去创建、销毁、读取和写入对象来被更新。交易必须将它希望操作的每个现有对象作为输入。此外，交易必须包括包对象的版本 *ID*、包内模块和函数的名称以及函数的参数（包括输入对象）。例如，调用函数：

```
public fun entrypoint(
  o1: Obj, o2: &mut Obj, o3: &Obj, x: u64, ctx: &mut TxContext
) { ... }
```

一个事务必须为三个不同的对象提供 ID，它们的类型是 Obj 和一个要绑定到 x 的整数。 TxContext 是由运行时填充的特殊参数，其中包含发送者地址和创建新对象所需的信息。 入口点的输入（更一般地，任何 Move 函数的输入）可以使用类型中编码的不同可变性权限进行传递。可以读取、写入、传输或销毁 Obj 输入。 &mut Obj 输入只能读取或写入，&Obj 只能读取。交易发送者必须被授权使用具有指定可变性权限的每个输入对象——有关更多详细信息，请参见第 4.4 节。

2.1.3 创建和传输对象。程序员可以通过使用传入入口点的 TxContext 为对象生成一个新的 ID 来创建对象：

```
public fun create_then_transfer(
  f: u64, g: OtherObj, o1: Obj, ctx: &mut TxContext
) {
  let o2 = Obj { id: TxContext::fresh_id(ctx), f, g };
  Transfer::transfer(o1, TxContext:sender());
  Transfer::transfer(o2, TxContext:sender());
}
```

此代码以 OtherObj 和 Obj 类型的两个对象作为输入，使用第一个对象和生成的 ID 创建一个新的 Obj，然后将这两个 Obj 对象传输给交易发送者。一旦一个对象被转移，它就会流入全局对象池，并且该交易的其余部分中不能被代码访问。Transfer 模块是 Sui 标准库的一部分，其中包括将对象传输到用户地址和其他对象的函数。

我们注意到，如果程序员代码忽略传输调用中的一个，则此代码将被 Move 类型系统拒绝。Move 强制执行资源安全 [5] 保护，以确保对象不能在未经许可的情况下创建、复制或意外破坏。资源安全的另一个例子是尝试传输同一个对象两次，这也将被 Move 类型系统拒绝。

3 SUI编程模型

在本节中，我们通过提供详细的语义定义来扩展第 2 节中对 Sui 编程模型的非正式描述。上一节展示了 Move 源代码的示例；这里我们定义了 Move 字节码的结构。 开发人员在自己的机器上编写、测试和形式验证 [10, 16] Move源代码，然后将其编译为Move字节码，然后再发布到区块链。任何在链上发布的 Move 字节码都必须通过字节码验证器[4, 5] 以确保其满足类型、内存和资源安全等关键属性。

如第 2 节所述，Move 是一种与平台无关的语言，无需分叉核心语言即可适应不同系统的特定需求。在以下描述中，我们定义了核心 Move 语言（用黑色文本表示）和扩展核心 Move 语言（用橙色文本表示）的 Sui 特定功能的两个概念。

3.1 模块

Module =	ModuleName×
	(StructName ⇀ StructDecl)×
	(FunName ⇀ FunDecl) × FunDecl
GenericParam =	[Ability]
StructDecl =	(FieldName ⇀ StorableType)×
	[Ability] × [GenericParam]
FunDecl =	[Type] [Type] × [Instr] × [GenericParam]
Instr =	TransferToAddr | TransferToObj | ShareMut |
	ShareImmut | . . .

Table 1: Module

代码被组织成模块，其结构在表 1 中定义。一个模块由一组命名结构声明和一组命名函数声明组成（这些声明的示例在第 2.1 节中提供）。一个模块还包含一个特殊的函数声明，用作模块初始化器。该函数在模块发布到链上时只调用一次。

结构声明是命名字段的集合，其中字段名称映射到可存储类型。它的声明还包括一个可选的*ability*列表（有关可存储类型和能力的描述，请参见第 2 节）。一个结构体声明还可以包含一个具有约束的泛型参数列表，在这种情况下，我们称其为泛型结构体声明，例如 struct Wrapper<T: copy>{ t: T }。泛型参数表示声明结构字段时要使用的类型——在声明结构时它是未知的，在实例化结构时提供具体类型（即，在创建结构值时）。

函数声明包括参数类型列表、返回类型列表和构成函数主体的指令列表。一个函数声明还可以包含一个具有能力约束的泛型参数列表，在这种情况下，我们称之为泛型函数声明，例如 fun unwrap<T: copy>(p: Wrapper<T>){}。与结构声明类似，泛型参数表示在函数声明时未知的类型，但在声明函数参数、返回值和函数体时仍使用该类型（调用函数时提供具体类型）。

可以出现在函数体中的指令包括所有普通的Move语言指令，除了全局存储指令（例如，move_to、move_from、borrow_global）。有关核心 Move 指令及其语义的完整列表，请参见 [14]。在Sui中， 持久存储是通过 Sui 的全局对象池而不是核心 Move 的基于帐户的全局存储来支持的。

有四种特定于 Sui 的对象操作。这些操作中的每一个都会更改对象的所有权元数据（参见第 3.3 节）并将其返回到全局对象池。最简单的是，一个 Sui 对象可以转移到一个 Sui 最终用户的地址。一个对象也可以转移到另一个父对象——这个操作需要调用者提供一个对父对象的可变引用以及子对象。一个对象可以可变地共享，因此它可以被 Sui 系统中的任何人读/写。最后，一个对象可以被不可变地共享，所以它可以被 Sui 系统中的任何人读取，但不能被任何人写入。

区分不同所有权的能力是sui的一个独特特征。在我们知道的其他区块链平台中，每个合约和对象都是可变共享的。正如我们将在第 4 节中解释的那样，Sui 将这些信息用于并行交易执行（对于所有交易）和并行协议（对于涉及没有共享可变性的对象的交易）。

3.2 类型和能力（ Types and Abilities ）

PrimType =	{address, id, bool, u8, u64, . . .}
StructType =	ModuleName × StructName× [StorableType]
StorableType =	PrimType ⊎ StructType⊎ GenericType ⊎ VectorType
VectorType =	StorableType
GenericType =	ℕ
MutabilityQual =	{mut, immut}
ReferenceType =	StorableType × MutabilityQual
Type =	ReferenceType ⊎ StorableType
Ability =	{key, store, copy, drop}

Table 2: Types and Abilities

Move 程序处理存储在 Sui 全局对象池中的数据和 Move 程序执行时创建的瞬态数据。对象和瞬态数据都是语言级别的Move值(变量）。然而，并非所有的值（变量）都是平等的——它们可能具有不同的属性和不同的结构，正如它们的类型所规定的那样。

Move 中使用的类型在表 2 中定义。Move 语言支持许多其他编程语言支持的基本类型，例如布尔类型或各种大小的无符号整数类型。此外，核心 Move 具有代表系统中最终用户的地址类型，该地址类型也用于识别交易的发送者和（在 Sui 中）对象的所有者。最后，Sui 定义了一个 id 类型，表示一个 Sui 对象的身份——详见第 3.3 节。

结构类型描述了在给定模块中声明的结构的实例（即值）（有关结构声明的信息，请参见第 3.1 节）。表示泛型结构声明的结构类型（即泛型结构类型）包括可存储类型的列表——该列表与结构声明中的泛型参数列表相对应。可存储类型可以是具体类型（原始类型或结构）或泛型类型。我们称此类类型为可存储的，因为它们可以作为结构的字段和持久存储在链上的对象出现，而引用类型则不能。

例如，Wrapper<u64> 结构类型是一个用具体（原始）可存储类型 u64 参数化的通用结构类型——这种类型可用于创建结构实例（即值）。另一方面，相同的泛型结构类型可以使用来自封闭结构或函数声明的泛型参数的泛型类型（例如，struct Parent<T> { w: Wrapper<T> }）进行参数化——这些种类的类型可用于声明结构字段、函数参数等。在结构上，泛型类型是在封闭结构或函数声明中泛型参数列表中的整数索引（在表 5 中定义为 N）。

Move 中的向量类型描述了同质值的可变长度集合。一个 Move 向量只能包含可存储类型，它本身也是可存储类型。Move 程序可以直接对值进行操作，也可以通过引用间接访问它们。引用类型包括引用的可存储类型和可变性限定符，用于确定（和强制执行）给定类型的值是可以读取和写入（mut）还是只能读取（immut）。因此，Move 值类型（表 2 中的类型）的最一般形式可以是可存储类型或引用类型。

最后，Move 中的“能力”控制给定类型的值允许哪些操作，例如是否可以复制给定类型的值。能力约束结构声明和泛型类型参数。 Move字节码验证器负责确保复制等敏感操作只能在具有相应能力的类型上执行。

3.3 对象和所有权（Objects and Ownership ）

TxDigest =	$Com(\text{Tx})$
ObjID =	$Com(\text{TxDigest} \times \mathbb{N})$
SingleOwner =	$\text{Addr} \uplus \text{ObjID}$
Shared =	$\{\texttt{shared\_mut}, \texttt{shared\_immut}\}$
Ownership =	$\text{SingleOwner} \uplus \text{Shared}$
StructObj =	$\text{StructType} \times \text{Struct}$
ObjContents =	$\text{StructObj} \uplus \text{Package}$
Obj =	$\text{ObjContents} \times \text{ObjID} \times \text{Ownership} \times \text{Version}$

Table 3: Objects and Ownership

此 ID 由创建它的交易分配给对象。对象 ID 是通过对当前交易的内容和记录交易创建了多少对象的计数器应用抗冲突哈希函数来创建的。由于对交易的输入对象的限制，交易（以及它的摘要）保证是唯一的，我们将在后面解释。

除了 ID 之外，每个对象还携带有关其所有权的元数据。一个对象要么由一个地址或另一个对象唯一拥有，要么以写/读权限共享，要么仅以读权限共享。对象的所有权决定了交易是否以及如何将其用作输入。从广义上讲，唯一拥有的对象只能用于由其所有者发起的交易或将其父对象作为输入，而共享对象可以由任何事务使用，但只能具有指定的可变性权限。 有关完整说明，请参见第 4.4 节。

有两种类型的对象：包代码对象和结构数据对象。一个包对象包含一个模块列表。结构对象包含一个 Move 结构值和该值的 Move 类型。对象的内容可能会改变，但其 ID、对象类型（包与结构）和Move的结构类型是不可变的。这确保了对象是强类型的并且具有持久的标识。

最后，一个对象包含一个版本。 新创建的对象的版本为 0，每次交易将对象作为输入时，对象的版本就会递增。

3.4 地址和验证器

Authenticator =	Ed25519PubKey ⊎ ECDSAPubKey ⊎ . . .
Addr =	*Com*(Authenticator)

Table 4: Addresses and Authenticators

地址是 Sui用户的永久身份（尽管请注意，单个用户可以拥有任意数量的地址）。要将对象传输给另一个用户，发送者必须知道接收者的地址。正如我们稍后将讨论的那样，Sui 交易必须包含发送（即发起）交易的用户的地址以及其摘要与地址匹配的身份验证器。地址和身份验证器之间的分离实现了加密敏捷性。验证器可以是来自任何签名方案的公钥，即使这些方案使用不同的密钥长度（例如，支持后量子签名）。此外，身份验证器不必是单个公钥——它也可以是（例如）一个 K-of-N 多重签名密钥。

3.5 交易（Transactions）

ObjRef =	ObjID × Version × Com(Obj)
CallTarget =	ObjRef × ModuleName × FunName
CallArg =	ObjRef ⊎ ObjID ⊎ PrimType
Package =	[Module]
Publish =	Package × [ObjRef]
Call =	CallTarget × [StorableType] × [CallArg]
GasInfo =	ObjRef × MaxGas × BaseFee × Tip
Tx =	(Call ⊎ Publish) × GasInfo × Addr × Authenticator

Table 5: Transactions

Sui 有两种不同的交易类型：发布新的 Move 包和调用之前发布的 Move 包。发布包含一个软件包的交易——一组将作为单个对象一起发布的模块，以及此包中所有模块的依赖关系（编码为对象引用列表，必须引用已发布的包对象 ）。为了执行发布的交易，Sui 运行时将在每个包上运行Move字节码验证器，将包与其依赖项链接起来，并运行每个模块的模块初始化程序。 模块初始化器对于引导由包实现的应用程序的初始状态很有用。

调用交易最重要的参数是对象输入。对象参数通过对象引用（对于单一所有者和共享不可变对象）或对象 ID（对于共享可变对象）指定。对于对象引用，运行时将根据池中对象的版本检查引用的版本，并检查引用的哈希是否与池对象匹配。这确保了运行时对象的视图与交易发送者的对象视图相匹配。

此外，调用交易接受类型参数和纯值参数。纯值可以包括原始类型和原始类型的向量，但不包括结构类型。

调用要调用的函数通过对象引用（必须引用包对象）、该包中的模块名称和该包中的函数名称来指定。为了执行调用交易，Sui 运行时将解析函数，将类型、对象和值参数绑定到函数参数，并使用 Move VM 执行函数。

调用和发布交易均需支付燃气计量和燃气费。计量限制由最大气体预算表示。运行时将执行事务，直到达到预算，如果预算用尽，将中止（除了扣除费用和报告中止代码）。[1]

费用从指定为对象参考的气体对象中扣除。该对象必须是 Sui 原生令牌（即，其类型必须是 Coin<SUI>）。Sui 使用 EIP1559[4] 风格的费用：协议定义了一个基本费用

4 https://github.com/ethereum/EIPs/blob/master/EIPS/eip- 1559.md

（以每个 Sui 代币的 gas 单位计价），该费用在 epoch 边界进行算法调整，并且交易发送者还可以包括一个可选的小费（以 Sui 代币计价）。在正常系统负载下，即使没有小费，交易也会得到及时处理。 但是，如果系统拥塞，小费较大的交易将被优先处理。 从 gas 对象中扣除的总费用为 (GasUsed ∗ BaseFee) + Tip。

3.6 交易结果

Event =	StructType × Struct
Create =	Obj
Update =	Obj
Wrap =	ObjID × Version
Delete =	ObjID × Version
ObjEffect =	Create ⊎ Update ⊎ Wrap ⊎ Delete
AbortCode =	ℕ × ModuleName
SuccessEffects =	[ObjEffect] × [Event]
AbortEffects =	AbortCode
TxEffects =	SuccessEffects ⊎ AbortEffects

Table 6: Transaction Effects

交易执行产生的交易结果在交易执行成功（表 6 中的 SuccessEffects）和不成功（表 6 中的 AbortEffects）的情况下是不同的。

成功执行交易后，交易影响包括对 Sui 的全局对象池所做更改的信息（包括对现有对象和新创建的对象的更新）和交易执行期间生成的事件。成功执行交易的另一个影响可能是从全局池中删除（即删除）对象，以及将一个对象包装（即嵌入）到另一个对象中，这与删除具有相似的效果——被包装的对象从全局池中消失 并且仅作为包装它的对象的一部分存在。由于已删除和包装的对象在全局池中不再可访问，因此这些效果由对象的 ID 和版本表示。

事件对成功执行事务的副作用进行编码，而不是对全局对象池的更新。在结构上，事件由 Move 结构及其类型组成。事件旨在由区块链之外的参与者使用，但不能被 Move 程序读取。

Move 中的交易具有全部执行或全部不执行的原子语义——如果交易的执行在某个时间点中止（例如，由于意外失败），即使对象发生了一些更改（或生成了一些事件）) 在此之前，这些影响都不会在中止的交易中持续存在。相反，中止交易效果包括数字中止代码和发生在交易中止的模块的名称。 中止的交易仍会收取 Gas 费。

4 SUI 系统

在本节中，我们从系统的角度描述 Sui，包括在拜占庭式失败的情况下确保权限管理者安全和活跃性的机制。我们还解释了客户端的操作，包括需要对系统状态进行一些保证而不验证其完整状态的轻客户端。

简要背景：在系统级别上，Sui 是 FastPay [3] 低延迟结算系统的演变，通过用户定义的智能合约扩展为对任意对象进行操作，并具有未经许可的委托权益委员会组成证明 [2]。对象所有者的基本资产管理基于拜占庭一致广播 [6] 的变体，与拜占庭共识的传统实现相比 [8,11,12]，它具有更低的延迟并且更容易在许多机器上进行扩展。当需要完全一致时，我们使用基于高通量 DAG 的共识，例如 [9] 来管理锁，同时在不同的共享对象上并行执行。

协议大纲：图 1 说明了客户端和 Sui 权限管理者之间的高级交互以提交事务。 我们在这里简要描述它们：

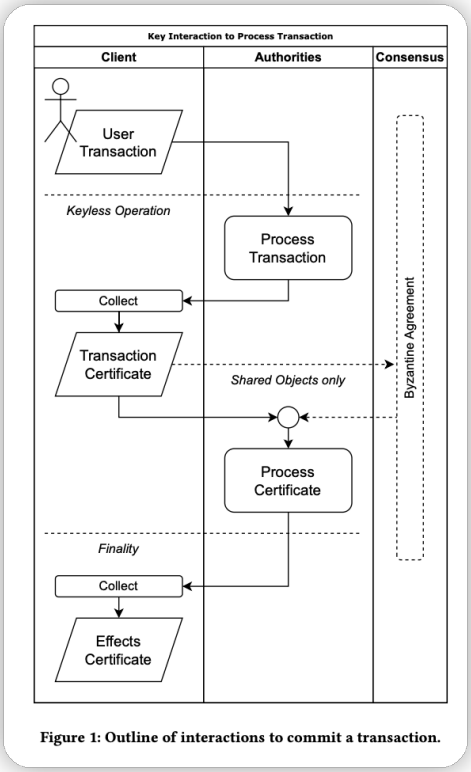

Figure 1: Outline of interactions to commit a transaction.

• 具有私有签名密钥的用户创建和签署用户事务以改变他们在 Sui 中拥有的对象或共享对象。. 随后，不再需要用户签名密钥，剩下的流程可以由用户客户端执行，也可以由网关代表用户执行（图中表示为无密钥操作）。

• 用户事务发送给 Sui 权限管理者，每个权限管理者检查它的有效性，并在成功后对其进行签名并将签名的事务返回给客户端。 客户端从法定人数中收集响应以形成交易证书。

• 然后将交易证书发送回所有权限管理者，如果交易涉及共享对象，它也会发送到由 Sui 权限管理者操作的拜占庭协议协议。 当局检查证书，如果涉及共享对象，还等待协议协议将其与其他共享对象事务相关的排序，然后执行事务并将其效果汇总为签名的效果响应。

• 一旦权限管理者的法定人数执行了该证书，其效果就是最终的（在图中表示为最终性）。 客户可以收集法定人数的权威响应并创建效果证书，并将其用作交易效果最终确定性的证明。

本节详细描述了这些操作中的每一个，以及跨越权限管理者重新配置和管理状态的操作。

4.1 系统模型(System Model)

Sui 运行在被表示为$e \in \{0, \ldots\}$一系列时期里。每个时期由一个委员会管理$Ce = (Ve, Se(\cdot))$，其中Ve是一组具有已知公共验证密钥和网络的权限的端点。函数$Se(v)$将每个权威$v \in Ve$映射到一个委托权益的单位数量。我们假设 Ce 对于每个epoch 由 epoch 的法定人数（见下文）签署e - 1.（第 4.7 节讨论委员会的组建和管理）。在一个时代内，一些权威是正确的（他们遵循协议忠实并且是实时的），而其他的是拜占庭（他

们任意偏离协议）。安全假设是诚实权威的集合 $He \subseteq Ve$ 被分配了法定人数在时代内。

$$\sum_{h \in H_e} S_e(h) > 2/3 \sum_{v \in V_e} S_e(v)$$

（并推荐任何一组拥有超过三分之二股份的权威机构作为法定人数）

在诚实的权威机构之间，至少存在一个有效且正确的一方作为每个证书的中继（参见第 4.3 节）。这确保了活跃性，并为拜占庭广播提供了最终的交付属性（参见 [6] 中的可靠广播的总体）。每个权限管理者单独或通过集体传播协议运行这样的中继。 包括 Sui 轻客户端、副本和服务在内的外部实体也可能担任此角色。包括 Sui 轻客户端、副本和服务在内的外部实体也可能担任此角色。 被动权威核心与不太可靠或可信的内部或外部主动中继组件之间的区别确保了对 Sui 的安全性和活跃性所依赖的可信计算库 [15] 的明确划分和最小化。

4.2 权威和副本数据结构（Authority & Replica Data Structures ）

sui权限管理者依靠许多数据结构来表示状态。 我们根据它们支持的操作来定义这些结构。 它们都具有确定性的字节表示。

一个对象（Obj）在 Sui 中存储用户智能合约和数据。 它们是 Sect 中介绍的 Move 对象的 Sui 系统级编码。 2. 它们支持以下一组操作：

• ref(Obj) 返回对象的引用(ObjRef)，即三元组（ObjID、Version、ObjDigest）。 ObjID 对于所有创建的新对象实际上是唯一的，而 Version 是一个递增的正整数，表示对象版本正在发生变化。

• owner(Obj) 返回对象所有者的身份验证器Auth。 在最简单的情况下，Auth 是一个地址，代表一个可以使用该对象的公钥。 还可以使用更复杂的身份验证器（参见第 4.4 节）。

• 如果对象是只读的，则只读（Obj）返回真。只读对象可能永远不会被变化、包装或删除。 它们也可能被任何人使用，而不仅仅是它们的所有者。

• parent(Obj) 返回上次改变或创建对象的事务摘要 (TxDigest)。

• contents(Obj) 返回对象类型 Type 和 data 可用于检查交易有效性和携带对象的应用程序特定信息的数据。

对象引用 (ObjRef) 用于索引对象。 它还用于验证对象，因为 ObjDigest 是对其全部内容的承诺。

交易(Tx) 是一种结构，表示一个或多个对象的状态转换。 它们支持以下一组操作：

• digest(Tx) 返回 TxDigest，它是对交易的绑定加密承诺。

• epoch(Tx) 返回可以执行此事务的 Epoch ID。

• inputs(Tx) 返回一个需要执行的交易对象 [ObjRef] 的序列。

• payment(Tx) 返回对用于支付 gas 的 ObjRef 的引用，以及最大 gas 限制，以及 gas 单位与 gas 支付对象中的价值单位之间的转换率。

•如果交易有效，valid(Tx,[Obj]) 返回真，给定提供的请求输入对象。 有效性在 Sect 中讨论。 4.4，并涉及授权对输入对象进行操作的交易，以及有足够的可用gas来支付其执行成本。

• exec(Tx,[Obj]) 执行事务并返回表示其效果的结构效果。 一个有效的交易执行是绝对可靠的，它的输出是确定性的。

交易由其 TxDigest 索引，也可用于验证其全部内容。所有有效的交易（除了特殊的硬编码创世交易）都有至少一个拥有的输入，即用于支付 gas 的对象。

交易结果（Effects）结构总结了交易执行的结果。 它支持以下操作：

• digest(Effects) 是对 Effects 结构的承诺 EffDigest，可用于索引或验证它。

• transaction(Effects) 返回执行的 TxDigest交易产生的影响。

• dependencies(Effects) 返回一系列依赖项[TxDigest] 应该在交易之前执行可以执行这些效果。

• contents(Effects) 返回执行摘要。状态报告智能合约执行的结果。 Created、Mutated、Wrapped、Unwrapped 和 Deleted 列表列出了经历了相应操作的对象引用。事件列出了执行发出的事件。

交易上的交易证书 TxCert 包含交易本身以及来自法定人数的标识符和签名。 请注意，证书可能不是唯一的，因为相同的逻辑证书可能由形成法定人数的不同权限集表示。此外，证书可能不会严格由 2/3 的法定人数签署，但如果有更多的权威机构响应，则可能更多。但是，同一事务中的两个不同的有效证书应被视为在语义上表示相同的证书。部分证书 (TxSign) 包含相同的信息，但来自一组代表权益的机构的签名低于所需的法定人数，通常是单个机构。签名者的标识符包含在证书中（即负责的签名 [?]），以识别准备处理证书的机构，或者可用于下载处理证书所需的过去信息（参见第 4.8 节）。

同样，结果证书 EffCert包含效果结构本身，以及来自权威机构的签名，代表交易有效时期的法定人数。 关于非唯一性和身份的相同警告适用于交易证书。 部分效果证书，通常包含单个授权签名，效果结构表示为 EffSign。

持久存储：每个权限管理者和副本都维护一组持久存储。 存储实现持久映射语义并且可以表示为一组键值对（表示为$map[key] \rightarrow value$），这样只有一对具有给定的键。在插入一对之前包含（key） call 返回 false，而 get(key) 返回错误。 插入一对之后， contains(key) 调用返回 true，get(key) 返回值。 权限管理者维护以下持久性存储：

• 订单锁映射 Lock$_v$ [ObjRef] → TxSignOption 记录由权威机构为拥有的对象版本 ObjRef 看到并签名的第一个有效交易 Tx，或者如果对象版本存在但没有有效交易用作输入，则记录已经看到 . 它还可以记录与该对象一起看到的第一个证书作为输入。 该表及其更新规则表示跨 Sui 权限的对象分布式锁的状态，并确保事务并发处理下的安全性。

• 证书映射 Ct$_v$ [TxDigest] → (TxCert, EffSign) 记录了由权威机构在其有效期内处理的所有完整证书 TxCert，其中还包括 Tx，以及它们的签名效果 EffSign。 它们由交易摘要 TxDigest 索引。

• 对象映射 Obj$_v$ [ObjRef] → Obj 记录了由 ObjRef 索引的 Ct$_v$ 内的证书中包含的交易创建的所有对象 Obj。 通过重新执行 Ct$_v$ 中的所

有证书，可以完全导出这个存储。 维护一个二级索引，将 ObjID 映射到具有此 ID 的最新对象。 这是处理新事务所需的唯一信息，维护旧版本只是为了方便读取和审计。

• 同步映射 Sync$_v$ [ObjRef] → TxDigest 将 Ct$_v$ 内的所有证书按它们创建、变异或删除的对象作为元组 ObjRef 进行索引。 通过处理 Ct$_v$ 中的所有证书可以完全重新创建此结构，并用于帮助客户端同步影响他们关心的对象的事务。

权限管理者维护所有四个结构，并提供对其证书映射的本地检查点的访问权限，以允许其他权威机构和副本下载其完整的已处理证书集。 副本不处理事务而只处理证书，并像当局一样重新执行它们以更新其他表。 它还维护一个订单锁映射以审计非模棱两可。

权限可以设计为维护所有四个存储（和检查点）以促进读取和同步的完整副本，并结合最小权限核心，该核心仅维护用于处理新对象的最新版本的对象锁和对象 交易和证书。 这最大限度地减少了安全所依赖的可信计算库。

只有顺序锁映射需要强键自一致性，即对键的读取应始终返回是否存在值或 None 对于存在的键，并且这种检查应该是原子的，并且更新将锁定设置为 非无值。 这是一个比键之间的强一致性更弱的属性，并且允许对存储进行有效的分片以进行扩展。 其他商店可能最终一致而不影响安全。

4.3 权限管理者的基本操作

交易处理。在收到交易 Tx 授权执行多项检查:

（1）它确保 epoch(Tx) 是当前 epoch。

（2） 它确保所有对象引用输入（Tx）和气体对象

payment(Tx) 中的引用存在于 Obj$_v$ 中，并将它们加载到 [Obj] 中。 对于拥有的对象，应该有确切的参考； 对于只读或共享对象，对象 ID 应该存在。

（3）确保可以在气体对象中提供足够的气体来支付执行交易的成本。

（4）它检查 valid(Tx, [Obj]) 是否为真。 此步骤确保事务中的身份验证信息允许访问拥有的对象。

（5） 它检查所有拥有的输入（Tx）对象的 Lockυ [ObjRef] 是否存在，它要么是 None 要么设置为相同的 Tx，并自动将其设置为 TxSign。 （我们称这些为“锁检查”）。

如果存在任何一种检查失败，则处理结束，并返回错误。 但是，Lock υ 的部分更新持续存在是安全的（尽管我们当前的实现不做部分更新，而是对所有锁进行原子更新）。

如果所有检查都成功，则权限管理返回交易签名，即。 部分证书 TxSign。 成功处理订单是幂等的，并返回部分证书 (TxSign) 或完整证书 (TxCert)（如果可用）。

任何一方都可以为一组权限核对交易和签名（TxSign），形成一个纪元e的法定人数，以形成交易证书 TxCert。

处理证书：收到证书后，权限管理者会检查交易的所有有效性条件，除了与锁相关的条件（所谓的“锁检查”）。相反，它执行以下检查：对于 input(Tx) 中的每个拥有的输入对象，它检查锁是否存在，以及它是否为 None、设置为任何 TxSign 或设置为与当前交易相同的证书证书。如果此修改后的锁检查失败，则权限管理者检测到不可恢复的拜占庭故障，停止正常操作，并启动灾难恢复过程。对于共享对象（参见第 4.4 节），当局检查锁定是否已通过在共识中排序的证书设置，以确定要使用的共享对象的版本。 如果是这样，交易可能会被执行；否则需要先等待这样的排序。

如果检查成功，则授权将证书添加到其证书映射中，连同其执行产生的效果，即。 C$t\upsilon$ [TxDigest] → (TxCert, EffSign); 它更新锁映射以记录证书 Lockυ [ObjRef] → TxCert 用于所有拥有的未设置为证书的锁的输入对象。 一旦 Input(Tx) 中的所有对象都插入到 Objυ 中，那么

EffSign 中的所有效果也会通过将它们的 ObjRef 和内容添加到 Obj$_v$ 来实现。 最后，对于在 EffSign 中创建或变异的所有内容，同步映射会更新以将它们映射到 Tx。

备注（remarks） 。处理交易和证书的逻辑导致了许多重要的属性：

• 因果关系和并行性。 交易和证书的处理条件都确保了因果执行：如果授权机构已经处理了创建交易所依赖的对象的所有证书（包括拥有的、共享的和只读的），那么它只会通过签署交易来“投票”。类似地，只有当它所依赖的所有输入对象都存在于它的本地对象映射中时，一个权威才会处理一个证书。 这强加了因果执行顺序，但也使不存在因果关系的事务能够在不同的内核或机器上并行执行。

• 签名一次，安全。 Lock$_v$ [·] 中所有拥有的输入对象锁被设置为使用它们通过检查的第一个交易 Tx，然后是使用该对象作为输入的第一个证书。 我们称之为将对象锁定到该事务，并且在一个时期内没有解锁。 因此，一个权限只为每个锁签署一个事务，这是一致广播 [6] 的重要组成部分，因此也是 Sui 的安全性。

• 灾难恢复。 一个权威机构检测到同一个锁的两个相互矛盾的证书，就有了不可恢复的拜占庭行为的证据——即证明法定人数诚实权威假设不成立。 这两个相互矛盾的证书是欺诈证明 [1]，可以与所有权限和副本共享以触发灾难恢复过程。 当局还可以获得其他形式的不可恢复的拜占庭行为证明，例如 >1/3 的签名效果 (EffSign)，表示证书执行不正确。 或者带有不代表先前处理的证书的正确输出的输入对象的证书。 这些也可以打包为欺诈证明，并与所有权威机构和副本共享。请注意，这些与证明少数权威（≤ 1/3 的权益）或对象所有者（任意数量）是拜占庭式或模棱两可的证明不同，这可以在不中断任何服务的情况下被容忍。

• 终局性。 对于 Lock$_v$ 、 Ct$_v$ 和 Obj$_v$ 、 Sync$_v$ 中的索引的任何读取请求，当局都会返回证书 (TxCert) 和签名效果 (EffSign)。 如果超过法定人数的授权报告 Tx 包含在其 Ct$_v$ 存储中，则交易被视为最终交

易。 这意味着效果证书 (EffCert) 是可转让的确定性证明。 但是，使用对象的证书也证明其因果路径中的所有依赖证书也是最终的。 向任何一方提供证书，然后该方可以将其提交给绝大多数机构进行处理，这也确保了证书效果的最终确定性。 请注意，finality 晚于 fastpay [3] 以确保重新配置下的安全性。 但是，权威机构可以在看到证书时应用事务的效果，而不是等待提交。

4.4 所有者、授权和共享对象

交易有效性（参见第 4.3 节）确保交易被授权在交易中包含所有指定的输入对象。 此检查取决于对象的性质以及所有者字段。

只读对象不能被改变或删除，并且可以在交易中同时被所有用户使用。 例如，移动模块是只读的。 这些对象确实有一个所有者，可以用作智能合约的一部分，但这并不影响使用它们的授权。 它们可以包含在任何交易中。

拥有的对象有一个所有者字段。 可以将所有者设置为代表公钥的地址。 在这种情况下，如果交易由该地址签名，则授权交易使用该对象并对其进行变异。 交易由单个地址签名，因此可以使用该地址拥有的一个或多个对象。 但是，单个事务不能使用由多个地址拥有的对象。 一个对象（称为子对象）的所有者可以设置为另一个对象（称为父对象）的 ObjID。 在这种情况下，只有当父对象包含在事务中并且事务被授权使用该对象时，才可以使用子对象。 合约可以使用此工具来构建有效的集合和其他复杂的数据结构。

共享对象是可变的，但没有特定的所有者。相反，它们可以包含在不同方的交易中，并且不需要任何授权。相反，他们执行自己的授权逻辑。这样的对象，由于必须在确保安全性和活跃性的同时支持多个写入者，确实需要完整的协议协议才能安全使用。因此，它们在执行之前需要额外的逻辑。当局按照 Sect 中的规定处理交易。 4.3 为拥有对象和只读对象管理它们的锁。然而，权威并不依赖于一致的广播来管理共享对象的锁。相反，涉及共享对象的交易的创建者将交易上的证书插入到高吞吐量的共识系统中，例如[9]。所有的权威机构都遵守这些证书的一致顺序，并根据该顺序分配每个事务使用的共享对象

的版本。然后可以继续执行，并保证在所有当局之间保持一致。权限包括在效果证书中的事务执行中使用的共享对象的版本。

上述规则确保了涉及只读和拥有对象的交易的执行只需要一致的广播和单个证书即可进行； 只有涉及共享对象的交易才需要拜占庭协议。 因此，智能合约作者可以设计他们的类型和他们的操作来优化对单个用户对象的传输和其他操作，以降低延迟，同时享受使用共享对象来实现需要由多个用户访问的逻辑的灵活性。

4.5 客户端

完整的客户端和副本。 副本，有时也称为完整客户端，不验证新交易，但出于审计目的维护系统有效状态的一致副本，以及构建交易或运营服务，包括。 读取轻客户端查询的基础设施。

轻客户端。 对象引用和事务都包含允许对导致它们创建或执行的事务的完整因果链进行身份验证的信息。 具体来说，对象引用 (ObjRef) 包含一个 ObjDigest，它是对象完整状态的验证器，包括获取父对象 (Obj) 的工具，即创建对象的 TxDigest。 类似地，TxDigest 验证交易，包括通过输入（Tx）提取输入对象的对象引用的工具。 因此，对象和证书的集合形成了一个自认证的二分图。 此外，效果结构也被签名，并且可以被整理成直接证明交易执行结果的效果证书。

这些设施可用于支持轻客户端，这些客户端可以对 Sui 状态执行高完整性读取，而无需维护完整的副本节点。 具体来说，权威或全节点可以提供简洁的证据包，包括交易 Tx 上的证书 TxCert 和对应于输入 (Tx) 的输入对象 [Obj]，以说服轻客户端可以在 Sui 内发生转换 . 然后，轻客户端可以提交此证书，或检查它是否已被法定人数或权威样本查看以确保最终确定性。 或者它可以使用执行产生的对象来制作交易，并观察它是否成功。

更直接地，服务可以向客户端提供效果证书，以使他们相信 Sui 中的过渡的存在和终结，而无需在系统内进行进一步的操作或交互。 如果最终证书的检查点可用，在纪元边界或其他地方，包括输入对象和证书的一束证据，以及在检查点中包含证书的证明也是最终性证明。

当局可以使用定期检查点机制来创建最终交易的集体检查点，以及随时间推移的 Sui 状态。 轻客户端可以使用在检查点上具有法定权益的证书来有效地验证对象的最近状态和发出的事件。 检查点机制对于时期之间的委员会重新配置是必要的。 更频繁的检查点对轻客户端很有用，也可以被当局用来压缩他们的内部数据结构，以及更有效地与其他当局同步他们的状态。

4.6 桥

对拜占庭协议管理的轻客户端和共享对象的原生支持允许 Sui 支持到其他区块链的双向桥接 [13]。 这种桥的信任假设反映了 Sui 和其他区块链的信任假设，如果其他区块链也支持轻客户端 [7]，则不必依赖可信预言机或硬件。

Bridges 用于导入在另一个区块链上发布的资产，以表示它并将其用作 Sui 系统中的封装资产。 最终，打包的资产可以被解锁并转移回本地区块链上的用户。 Bridges 还可以允许在 Sui 上发行的资产被锁定，并在其他区块链上用作包装资产。 最终，其他系统上的包装对象可以被销毁，并且 Sui 上的对象更新以反映状态或所有权的任何更改，并解锁。

桥接资产的语义对于确保包装资产有用具有一定的重要性。 跨区块链桥接的可替代资产可以提供更丰富的包装表示，允许它们在包装时可分割和可转移。 不可替代资产不可分割，而只能转让。 它们还可能支持在包装时以受控方式改变其状态的其他操作，这可能需要在它们被桥接和解包时执行自定义智能合约逻辑。 Sui 很灵活，允许智能合约作者定义此类体验，因为桥接器只是在 Move 中实现的智能合约，而不是原生的 Sui 概念——因此可以使用 Move 提供的可组合性和安全保证进行扩展。

4.7 委员会重组

当委员会 Ce 被委员会 Ce' 替换时，在 epoch 之间会发生重新配置，其中 $e' = e + 1$。重新配置安全性确保如果在 e 或更早之前提交了事务 Tx，则在 e 之后不能提交冲突事务。 Liveness 确保如果 Tx 在 e 或之前提交，那么它也必须在 e 之后提交。

我们利用 Sui 智能合约系统来执行重新配置所需的大量工作。 在 Sui 中，系统智能合约允许用户锁定并将权益委托给候选机构。 在一个时期内，硬币的所有者可以通过锁定代币自由委托、通过解锁代币取消委托或将其委托更改为一个或多个权限。

一旦 epoch 的权益达到法定人数 e 投票结束 epoch，当局就会交换信息以提交检查点，确定下一个委员会，并更改 epoch。 首先，当局在协议协议 [9] 的帮助下运行检查点协议，以在 epoch 结束时就经过认证的检查点达成一致。 检查点包含所有事务的联合，以及可能产生的对象，这些对象已由法定人数处理。 因此，如果一笔交易已由法定人数处理，那么至少有一个诚实的处理它的权威机构会将其处理过的交易包含在时代结束检查点中，从而确保交易及其影响在各个时代都是持久的。 此外，这样一个经过认证的检查点保证了所有交易都可供纪元e的诚实当局使用。然后使用在纪元结束检查点的权益委托来确定纪元 $e + 1$ 的新权限集。旧权限权益的法定人数和新权限权益的法定人数都签署了新委员会 Ce'， 以及新纪元开始的检查点。 一旦这两组签名都可用，新的权限集开始处理新纪元的交易，旧的权限可以删除它们的纪元签名密钥。

恢复。 由于客户端错误或客户端模棱两可，拥有的对象可能会在一个时期内被“锁定”，从而阻止任何有关它的交易被认证（或最终确定）。 例如，客户端使用相同的拥有对象版本签署两个不同的交易，每个交易有一半的授权机构签署，将无法形成

一个要求两个证书中的任何一个签名的法定人数的证书。 恢复确保一旦 epoch 发生变化，这些对象再次处于允许它们在事务中使用的状态。 由于无法形成证书，因此原始对象在下一个要操作的时期开始时可用。 由于交易包含一个纪元号，旧的模棱两可的交易不会再次锁定该对象，让其所有者有机会使用它。

奖励和加密经济学。 Sui 有一个原生代币 SUI，供应量固定。 SUI 用于支付 gas 费用，也可用作一个时期内授权的委托权益。 在这个时期内，权威的投票权是这个委托权益的函数。 在 epoch 结束时，通过所有处理的交易收取的费用根据他们对 Sui 运营的贡献分配给当局，然后他们分享一些费用作为奖励给委托给他们的地址的地址。 我们将 Sui 的代币经济学的完整描述推迟到专门的论文中。

4.8 权威和副本更新

客户驱动。 由于客户端故障或非拜占庭授权失败，某些授权可能未处理所有证书。 因此，依赖于这些证书生成的缺失对象的因果相关交易将被拒绝。 但是，客户始终可以将诚实的权威更新到能够处理正确交易的程度。 它可以使用自己的过去证书存储，或使用一个或多个其他诚实的权威机构作为过去证书的来源来做到这一点。

给定一个证书 c 和一个包含 c 及其因果历史的 Ctv 存储，客户端可以将诚实的权威 v' 更新到也可以应用 c 的程度。 这涉及找到不在 v' 中的最小证书集，以便在应用时 v' 中的对象包括 c 的所有输入。 使用包含证书 TxCert 的商店 Ctv 更新滞后的权威机构 B 涉及：

• 客户端维护一个要同步的证书列表，最初设置为仅包含 TxCert。

• 客户端认为最后一个需要同步的 TxCert。 它提取 TxCert 中的 Tx 并派生其所有输入对象（使用 Input(Tx)）。

• 对于每个输入对象，它检查最后生成或变异的 Tx（使用 Ctv 上的 Sync$v$ 索引）是否在 B 中有证书，否则从Ctv 读取其证书并将其添加到要同步的证书列表中。

• 如果没有更多证书可以添加到列表中（因为 B 中缺少更多输入），则证书列表按因果顺序排序并提交给 B。

上述算法也适用于将对象更新到特定版本以启用新事务。 在这种情况下，生成对象版本的 Tx 的证书（在 Sync$v$ [ObjRef] 中找到）被提交给滞后机构。 一旦在 B 上执行，正确版本的对象就可以使用。

执行此操作的客户端称为中继器。 可以有多个中继器独立并同时运行。 它们在完整性方面不受信任，并且它们的操作是无密钥的。 除了客户端，权限可以运行中继器逻辑来相互更新，副本操作服务也可以充当中继器来更新滞后的权限。

块。 当局为追随者在处理证书时接收更新提供了便利。 这允许副本维护权威状态的最新视图。 此外，当局可以使用推拉式八卦网络在短期内相互更新最新处理的交易，并减少中继器执行此功能的需要。 从长远来看，滞后的当局可能会在时代边界或更频繁地使用定期状态承诺，以确保他们已经处理了直到某些检查点的完整证书集。

5 扩展和延迟

Sui 系统允许通过将更多资源（即一台机器内或多台机器上的 CPU、内存、网络和存储）用于处理事务的权限进行扩展。 更多的资源会提高处理交易的能力，从而增加为这些资源提供资金的费用收入。 更多的资源也会导致更低的延迟，因为无需等待必要的资源可用就可以执行操作。

吞吐量。 为了确保更多的资源以准线性方式增加容量，Sui 设计积极减少了瓶颈和需要在权限内进行全局锁定的同步点。 处理事务被明确地分为两个阶段，即（1）确保事务对特定版本的拥有或共享对象具有独占访问权，以及（2）随后执行事务并提交其效果。

阶段 (1) 要求事务以对象的粒度获取分布式锁。 对于拥有的对象，这是通过可靠的广播原语执行的，不需要权限内的全局同步，因此可以通过 ObjID 跨多台机器的锁管理分片来扩展。 对于涉及共享对象的事务，需要使用共识协议进行排序，该协议确实对这些事务施加了全局顺序，并且有可能成为瓶颈。 然而，工程高吞吐量共识协议 [9] 的最新进展表明，顺序执行是状态机复制的瓶颈，而不是排序。 在 Sui 中，排序仅用于确定输入共享对象的版本，即递增对象版本号并将其与事务摘要相关联，而不是执行顺序执行。

阶段 (2) 发生在所有输入对象的版本为权威所知（并且跨权威安全地同意）并且涉及执行移动交易和承诺其效果时。 一旦知道输入对象的版本，就可以完全并行执行。 移动多核或物理机上的虚拟机读取版本化的输入对象，执行并将生成的对象写入存储。 对象和事务存储的一致性要求（除了顺序锁映射）非常宽松，允许每个权限内部使用可扩展的分布式键值存储。 执行是幂等的，即使处理执行的组件发生崩溃或硬件故障也很容易恢复。

因此，彼此没有因果关系的事务的执行可以并行进行。 因此，智能合约设计者可以在其合约中设计对象和操作的数据模型，以利用这种并行性。

检查点和状态承诺是根据关键事务处理路径计算的，不会阻止新事务的处理。 这些涉及对已提交数据的读取操作，而不是在事务达到最终确定之前需要计算和协议。 因此，它们不会影响处理新事务的延迟或吞吐量，并且它们本身可以分布在可用资源中。

读取可以从非常积极的、可扩展的缓存中受益。 权威机构签署并提供轻客户端读取所需的所有数据，这些数据可以由分布式存储作为静态数据提供。 证书作为交易

和对象的完整因果历史的信任根。 状态承诺进一步允许整个系统对所有状态和处理的交易具有定期的全球信任根，至少在每个时期或更频繁地处理。

潜伏。 智能合约设计者可以灵活地控制他们定义的操作的延迟，这取决于它们是否涉及拥有或共享的对象。 拥有的对象在执行和提交之前依赖于可靠的广播，这需要两次往返到法定人数才能达到最终性。 另一方面，涉及共享对象的操作需要一致的广播来创建证书，然后在共识协议中进行处理，从而导致延迟增加（截至 [9] 需要 4 到 8 次往返仲裁）。

REFERENCES

[1] Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, and Ismail Khoffi. 2021. Fraud and Data Availability Proofs: Detecting Invalid Blocks in Light Clients. In Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part II (Lecture Notes in Computer Science, Vol. 12675), Nikita Borisov and Claudia Diaz (Eds.). Springer, 279–298.

[2] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick Mc- Corry, Sarah Meiklejohn, and George Danezis. 2019. SoK: Consensus in the Age of Blockchains. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019. ACM, 183–198.

[3] Mathieu Baudet, George Danezis, and Alberto Sonnino. 2020. FastPay: High- Performance Byzantine Fault Tolerant Settlement. In AFT '20: 2nd ACM Confer- ence on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020. ACM, 163–177.

[4] Sam Blackshear, Evan Cheng, David L. Dill, Victor Gao, Ben Maurer, Todd Nowacki, Alistair Pott, Shaz Qadeer, Ra in, Dario Russi, Stephane Sezer, Tim Za- kian, and Runtian Zhou. 2019. Move: A Language With Programmable Resources. https://developers.libra.org/docs/move- paper.

[5] Sam Blackshear, David L. Dill, Shaz Qadeer, Clark W. Barrett, John C. Mitchell, Oded Padon, and Yoni Zohar. 2020. Resources: A Safe Language Abstraction for Money. CoRR abs/2004.05106 (2020). arXiv:2004.05106 https:// arxiv.org/abs/ 2004.05106

[6] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. 2011. Introduction to reliable and secure distributed programming. Springer Science & Business Media.

[7] Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. 2021. SoK: Blockchain Light Clients. IACR Cryptol. ePrint Arch. (2021), 1657.

[8] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, An- drei Tonkikh, and Athanasios Xygkis. 2020. Online Payments by Merely Broad- casting Messages. In 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2020, Valencia, Spain, June 29 - July 2, 2020. IEEE, 26–38.

[9] George Danezis, Eleftherios Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2021. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus. CoRR abs/2105.11827 (2021).

[10] David L. Dill, Wolfgang Grieskamp, Junkil Park, Shaz Qadeer, Meng Xu, and Jingyi Emma Zhong. 2021. Fast and Reliable Formal Verification of Smart Con- tracts with the Move Prover. CoRR abs/2110.08362 (2021). arXiv:2110.08362 https://arxiv.org/abs/2110.08362

[11] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos- Adrian Seredinschi. 2018. AT2: Asynchronous Trustworthy Transfers. CoRR abs/1812.10844 (2018).

[12] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos- Adrian Seredinschi. 2019. The Consensus Number of a Cryptocurrency. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019, Peter Robinson and Faith Ellen (Eds.). ACM, 307–316.

[13] Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. 2021. SoK: Validating Bridges as a Scaling Solution for Blockchains. IACR Cryptol. ePrint Arch. (2021), 1589.

[14] Marco Patrignani and Sam Blackshear. 2021. Robust Safety for Move. CoRR abs/2110.05043 (2021). arXiv:2110.05043 https://arxiv.org/abs/2110.05043

[15] Jerome H Saltzer and Michael D Schroeder. 1975. The protection of information

in computer systems. Proc. IEEE 63, 9 (1975), 1278–1308.

[16] Jingyi Emma Zhong, Kevin Cheang, Shaz Qadeer, Wolfgang Grieskamp, Sam

Blackshear, Junkil Park, Yoni Zohar, Clark W. Barrett, and David L. Dill. 2020. The Move Prover. In Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12224), Shuvendu K. Lahiri and Chao Wang (Eds.). Springer, 137–150. https://doi.org/10.1007/978- 3- 030- 53288- 8_7

[10] David L. Dill, Wolfgang Grieskamp, Junkil Park, Shaz Qadeer, Meng Xu, and Jingyi Emma Zhong. 2021. Fast and Reliable Formal Verification of Smart Con- tracts with the Move Prover. CoRR abs/2110.08362 (2021). arXiv:2110.08362 https://arxiv.org/abs/2110.08362

[11] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos- Adrian Seredinschi. 2018. AT2: Asynchronous Trustworthy Transfers. CoRR abs/1812.10844 (2018).

[12] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos- Adrian Seredinschi. 2019. The Consensus Number of a Cryptocurrency. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019, Peter Robinson and Faith Ellen (Eds.). ACM, 307–316.

[13] Patrick McCorry, Chris Buckland, Bennet Yee, and Dawn Song. 2021. SoK: Validating Bridges as a Scaling Solution for Blockchains. IACR Cryptol. ePrint Arch. (2021), 1589.

[14] Marco Patrignani and Sam Blackshear. 2021. Robust Safety for Move. CoRR abs/2110.05043 (2021). arXiv:2110.05043 https://arxiv.org/abs/2110.05043
[15] Jerome H Saltzer and Michael D Schroeder. 1975. The protection of information

in computer systems. Proc. IEEE 63, 9 (1975), 1278–1308.
[16] Jingyi Emma Zhong, Kevin Cheang, Shaz Qadeer, Wolfgang Grieskamp, Sam

Blackshear, Junkil Park, Yoni Zohar, Clark W. Barrett, and David L. Dill. 2020. The Move Prover. In Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12224), Shuvendu K. Lahiri and Chao Wang (Eds.). Springer, 137–150. https://doi.org/10.1007/978- 3- 030- 53288- 8_7