WebAssembly 在区块链中的实践

蚂蚁链平台技术部

张磊

区块链

- 分布式账本
  - 去中心化的点对点网络
  - 每个节点保存相同的账本数据
- 区块链 1.0
  - 简单的记账功能，可以实现数字货币
  - 代表：比特币
- 第二代区块链
  - 支持**智能合约**，可以实现复杂的分布式应用
  - 代表：以太坊

智能合约

- 运行在区块链上的一段用户程序
  - **分布式**：在区块链网络的每一个节点上执行
  - **确定性**：在所有节点上的执行结果完全一致
  - **安全性**：不能影响区块链节点的正常工作
  - **图灵完备**：可以表达复杂的业务逻辑
- 编程模型与普通程序有较大差别
  - 不支持任何系统调用 => 无法保证执行结果确定性
  - 计算资源受严格监控 => 不适合计算密集型任务

智能合约 - Gas 计费

- 图灵完备 => 停机问题
  - 无法判定智能合约能否终止执行 => 死循环/死递归
  - 消耗所有区块链节点的计算资源 => DDoS 攻击
- 超时检查无法保证确定性
- 需要精确度量合约执行过程中的资源消耗
  - 每一条指令
  - 每一次内存访问
  - 每一次存储访问
- Gas 耗尽时立即终止智能合约执行

智能合约

- 理想的执行环境：**虚拟机**
  - 提供安全隔离的沙盒环境
  - 保证执行结果的确定性
  - 精确监控运行时资源消耗
- 市面上的虚拟机大多无法满足要求
  - 功能过于复杂，无法保证安全性和确定性
    - 网络访问、文件访问、浮点数 等
  - 缺乏有效的运行时资源监控机制

以太坊

- 以太坊 1.0：EVM
  - 栈式字节码虚拟机（256 位字宽）
  - 提供访问区块链存储的专用指令
  - 性能差，支持高级语言少
    - 主流语言：Solidity
- 以太坊 2.0：ewasm
  - 标准 wasm 的子集
    - 不支持浮点数
  - 定义了 wasm 与区块链交互的接口

智能合约

```
// ERC20
function transfer(
    address sender,
    address recipient,
    uint256 amount
) public {
    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    uint256 senderBalance = _balances[sender];
    require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
    _balances[sender] = senderBalance - amount;
    _balances[recipient] += amount;

    emit Transfer(sender, recipient, amount);
}
```

智能合约

- 部署合约
  - 用户把合约代码编译成字节码，发起交易部署到区块链上
    - 每个合约定义一组可被调用的接口
  - 区块链检查字节码合法性，存储到每一个节点上
- 调用合约
  - 用户指定接口和参数，发起交易调用链上的合约
  - 每个区块链节点执行完全相同的合约逻辑
    - 执行过程中产生一些数据存储到区块链上
- Function as a Service?

Wasm x Blockchain

- 越来越多的区块链开始采用 wasm
  - 公链：EOS, Substrate, NEAR, etc.
  - 联盟链：蚂蚁链，长安链，百度 XuperChain，等
- 为什么选择 WebAssembly?
  - 安全性
  - 跨平台
  - 确定性
  - 高性能

安全性

- 沙箱环境
  - 内存安全：智能合约不能访问合约以外的内存空间
  - 控制流安全：智能合约只能调用区块链提供的特定接口
- Gas 计费
  - 对 wasm 指令进行 gas 度量
  - 准确性：wasm 指令语义与机器指令十分接近
  - 确定性：所有节点运行完全相同的指令序列
- 隐私保护
  - 可以运行在可信执行环境中（如 Intel SGX）

跨平台

- 区块链去中心化 => 复杂的节点部署环境
- 可能运行不同的操作系统
  - Linux, macOS, Windows
- 可能使用不同的处理器架构
  - x86-64, ARM, RISC-V
- 可能运行不同的客户端实现
  - C++，Rust，Go
- 智能合约需要运行在所有可能的节点环境上

确定性

- 相同的代码 + 相同的输入 => 相同的输出？
  - 智能合约在各个区块链节点的执行结果完全一致
- Wasm 执行模型中存在非确定性因素
  - 多线程：线程调度顺序不确定
    - 智能合约不支持多线程
  - 浮点数：CPU 浮点指令计算结果不确定
  - Host 函数：执行在 wasm 沙箱之外
    - 由区块链平台自己保证代码逻辑确定性
  - 资源耗尽：节点的堆/栈内存空间不确定

确定性 - 浮点数

- IEEE 754 未规定 NaN 严格二进制表示
  - Single float: s111 1111 1xxx xxxx xxxx xxxx xxxx xxxx
  - 不同处理器产生的 NaN 表示不完全一致
- Wasm SIMD 规范允许 subnormal flushing
  - 部分处理器厂商默认将 subnormal 值置零
- 解决方法
  - 智能合约中禁止使用浮点数
  - 使用整数指令模拟浮点数运算
  - 对硬件浮点指令计算结果规范化

高性能

- 解释器
  - 优点：实现简单，可移植性好
  - 缺点：性能差
- JIT/AOT
  - 优点：性能好，可接近 native
  - 缺点：实现复杂，需要为不同处理器架构开发后端
  - 缺点：编译时间较长，影响冷启动性能
- 单纯使用解释器无法达到“高性能”的目标
  - Chrome V8 默认不启用 wasm 解释器

高性能 - Gas 计费

- Gas 计费有显著的运行时开销
  - 检查当前剩余 gas + 扣除下一条指令 gas
- 以指令为单位
  - 执行每一条指令前加入 gas 判断逻辑
- 以基本块为单位
  - 执行每一个控制流基本块前加入 gas 判断逻辑
  - Wasm 的结构化控制流有利于基本块划分
- 硬件加速
  - 定制化处理器指令实现 gas 计费

高性能 && 确定性

- 解释器
  - 执行逻辑受运行环境（处理器，OS）影响较小
- JIT/AOT
  - 直接以机器码形态执行，受运行环境影响较大
  - 不同环境下的执行结果一致性难以保证
    - JIT 编译 O0 优化 vs O2 优化?
    - x86_64 JIT vs ARM JIT?
  - JIT bomb：针对 JIT 编译器的 DDoS 攻击
- **现状**：以太坊等主流公链仍以解释器为主

TEE - Intel SGX

- 硬件可信执行环境，保护链上用户数据隐私
- 智能合约虚拟机运行在可信区域中
  - 类似嵌入式环境，不支持 POSIX 系统调用
- JIT/AOT 虚拟机对系统调用依赖度较高
  - 分配可执行内存：mmap
  - 隐式内存边界检查：mprotect
  - 异常处理：signal handling
- **现状**：目前基于 SGX 的区块链平台以解释器为主

WAMR

- 原生支持 SGX
  - 核心 runtime 部分不依赖任何系统调用
  - Wasmtime, wasmer 等 JIT 虚拟机对 SGX 不友好
- 执行效率高
  - 利用 LLVM 的强大编译优化能力
  - 相比解释执行提升 10-20x
- 启动速度快
  - 智能合约运行周期短，对启动时间敏感
  - instance 创建：WAMR 7us, Wasmtime 30us, Wasmer 100us

未来：多语言支持

- 目前对 wasm 支持最好的大多是静态编译型语言
  - C/C++, Rust, Go
- 智能合约开发者对解释型/托管型语言更加熟悉
  - Java, Python, TypeScript
- 在 wasm 上支持托管型语言仍存在许多困难
  - 依赖 Interface Types, GC 等 post-MVP 功能
- Java => wasm?

未来：调试能力

- wasm 还没有十分成熟的调试功能解决方案
  - Source Map：栈帧不准确，无法查看变量
  - DWARF：格式复杂，实现难度较高
- 区块链智能合约调试本身具有挑战性
  - 执行逻辑依赖链上数据，难以在本地复现
  - 链上调试不能影响其他合约正常执行
- 目前合约调试主要依靠 print/log 大法

未来：高性能 + 确定性

- 对区块链更加友好的 wasm 虚拟机
  - 兼顾 JIT 的高性能和解释器的确定性
- 保证 JIT 与解释器的行为一致性
  - 支持 tiering：解释器 => baseline JIT => optimizing JIT
- 保证不同处理器上 JIT 行为一致性
  - 支持 ARM 和 x86-64 混合部署的区块链网络
- 轻量级 JIT 编译器
  - 编译速度快，可运行于 SGX 中

蚂蚁链 x WebAssembly

- 多语言
  - C++, Go, AssemblyScript
  - WIP：智能合约领域专用语言
- 工具
  - 智能合约集成开发环境（IDE）
  - 自研 wasm 智能合约调试器
- 高性能
  - 解释器与 AOT 双引擎
  - 定制化硬件 gas 计费指令
- 安全
  - 支持 Intel SGX 和国产 TEE 平台
  - 编译期/运行时软浮点等深度定制

欢迎加入蚂蚁链！

- 工作内容
  - wasm 虚拟机开发及性能优化
  - 高级语言到 wasm 的编译器开发
  - 基于 wasm 的调试功能开发
- 期望你
  - 熟悉编译原理，有 GCC, LLVM 等开源编译器开发经验
  - 熟悉虚拟机和编程语言 runtime 工作机制
  - 熟悉计算机体系结构，有软硬件协同优化经验
- 联系：shifei.zl@antgroup.com

谢谢！

微信扫码添加

WebAssembly Open Day 小助手

邀请你进 WAMR 社区开发者群

微信扫码关注

金融级分布式架构

参加“WebAssembly Open Day ”抽奖活动