Grug

Larry Engineer
Left Curve Software
larry@leftcurve.io

Initial version: May 24, 2024

Abstract

Grug is an execution environment for blockchains. It's an alternative to the EVM, Solana VM, Move, and CosmWasm.

1 Characteristics

Grug is much faster than single-threaded EVM, but probably not as fast as SVM or Move (they are the results of multiple years of grinding by extremely well-funded teams; we can't compete with that).

Instead, Grug differentiates itself by its **rich features**. The idea is that there're many apps that are difficult to build in EVM, SVM, Move, etc. because those VMs lack certain features that these apps need, but they will be easy to build with Grug. Such features include:

1.1 Smart accounts

While it took Ethereum almost 4 years to ship EIP-3074,[1] Grug comes packaged with a complete smart account solution from the get go.

This solution builds on our previous work of the very first smart account solution in Cosmos,[2] which is used by Xion[3] to onboard 100k users, processing 1100 TPS without any hiccup.

Our solution allows one to program smart contract accounts that authenticate transactions with arbitrary logics as the developer likes. This can include Passkey, one-time password, two-factor authentication, and so on.

We believe the endgame wallet experience is a key-less one. The user will, for example, generate and store a private key in their iCloud so that it's accessible from any device, and sign transactions with fingerprint and Face ID. Our smart account solution enables this.

Another use case uniquely enabled by our smart account solution is a new DeFi primitive, the **credit account**.[4] It allows users to take uncollateralized* margin loans, something not possible with traditonal lending protocols such as Compound and Aave. Credit accounts are also cross-margin, meaning one can take on multiple activities within the account: spot trading, futures trading, yield farming... which will collateralize each other (capital efficient).

Credit account is possible thanks to two innovations: 1) the account will backrun each transaction, rejecting it if the actions result in the health factor dropping below a threshold; and 2) a liquidation mechanism is built into the account. While it's not impossible to implement these in traditional VMs (see similar works by Mars[6] and Gearbox[7] protocols), they tend to suffer from high complexity (a lot of workarounds are necessary due to lack of a smart account system), poor user experience (e.g. credit accounts are not "real" accounts hence won't show up in wallet apps), and high gas cost. Grug can easily reduce the codebase size by 80%, be much cheaper, and offer good UX.

*The legally more rigorous term is lien.[5] The loan is in fact fully collateralized, it's just that the borrower still has fully control over the collateral. This is similar to how a car loan or a mortgage works.

1.2 Customizable gas fees

A guideline we employ in designing Grug is that *if we expect developers may want to customize something, then this thing should be abstracted into a smart contract; if it's something that we don't expect any developer to ever want to customize, then it will be native Rust code*. With this, developers never have to fork chain-level code to achieve their objectives. All they need to do is to write contracts.

This is in contrary to the EVM, in that, for example, if you don't like the floating gas price in EIP-1559 and want a flat price instead, you will have to fork and change the core Ethereum protocol. This is risky and beyond the capability of most developers.

In Grug, gas fees are managed by a smart contract, nicknamed the **taxman**. After each transaction, taxman is invoked and provided with a report of the transaction and its gas consumption. Developers can implement any logic to handle the fee, for example:

- burn the fee
- distribute the fee to stakers
- discounted gas price for holders of a certain NFT
- if the transaction is placing a limit order, then hold the fee; only charge it if the order is filled; refund if canceled

Additionally, if at any time the community wishes to change the fee mechanism, they don't have to harkfork the chain. All that's necessary is to migrate the contract.

Another interesting use case is that since taxman backruns every transaction, it's possible to use it to capture protocol-owned MEV, similar to Osmosis' ProtoRev[8] module.

1.3 Cronjobs

A cronjob is an operation that needs to be performed at regular time intervals. For instances,

- a perpetual futures protocol needs to update its funding rate once in a while based on long/short open interests
- an orderbook DEX needs to match buy and sell orders at the end of each block
- a dollar cost averaging (DCA) service needs to carry out the buy/sell operations for the user at regular intervals

In EVM, developers need to rely on bots to carry out these, since actions can only be triggered by submitting transactions. Relying on offline infrastrucutre such as a bot network increases operating cost and are susceptible to network congestions as discussed in the previous section.

The Cosmos SDK pioneered a solution to this known as the Begin/EndBlocker. Essentially, developers can create actions to be performed automatically at the beginning or the end of each block, without manual invocation. Since these actions are not triggered by transactions, they do not suffer from congestions.

In Grug, we borrow the idea of Begin/EndBlockers,[9] giving smart contracts access to them.

1.4 A new token standard

ERC-20 is a terrible token standard. It's bad for users, bad for developers, bad for indexers. Literally no one likes it, but Ethereum is stuck with it for eternity since numerous DeFi apps already depend on it.

A few reasons why ERC-20 is so bad:

- You can only transfer one token per transaction.
- It's not possible to send tokens to a contract then atomically call a method on that contract. This feature is termed transferAndCall and is available on several other token standards such as ERC-677, but they are not widely adopted; even if they are, you can still only transfer one token at a time, so for use cases where it's necessary to transfer two or more tokens then call the contract (e.g. providing liquidity to an AMM) this doesn't work.
- To solve the previous problem, the approval mechanism was invented. This however created massive security risks. Many apps have users make infinite approvals in order to save on gas, but this also means in case the app is exploited, users' funds can be infinitely stolen.

- The recipient can't reject a token transfer. There has been numerous cases where users mistakenly send tokens to a contract which are irreversibly lost. EVM allows contract to implement a receive function, in which the contract can reject transfers, but it only works for ETH, not ERC-20s.
- Given an account, it's impossible to find all ERC-20s it holds. This is because token balances aren't stored in a single location, but rather in thousands of separate contracts. Developers first need an archive node to parse all contract deployment history to build a list of all ERC-20s in existence, then make a query at each and every of them to find the user's balance. That's a ridiculous amount of work for such a basic task.
- Given an ERC-20, it's impossible to find all accounts that hold the token. Instead, developers need access to an archive node and index historical mint/burn/transfer events. The lack of this feature has its root in Ethereum's choice of Merkle Patricia Tree and thus isn't fixable.

Grug's native token standard fixes all these issues except for the last one (we can do it, but doing so increases state size a lot and as far as I'm concerned, isn't worth it). Furthermore, it unifies fungible tokens and NFTs under a single standard, borrowing ideas from Osmosis[10] and Metaplex.[11]

1.5 Cosmos IBC

IBC[12] is the best interoperability protocol (if you disagree, you're wrong). Grug enshrines IBC natively as part of the VM.

On the business side, having IBC allows us to easily onboard USDC holders from any ecosystem though Circle's CCTP[13] and Noble.[14]

1.6 More quality of life improvements for developers...

- **Multicall**: Whereas in EVM, each transaction can only make a single contract call, Grug allows you to compose as many actions into a single transaction (as long as they fit in the block gas limit).
- **Iteration**: In EVM, it is not possible to iterate data in a mapping data structure. The lack of this capability means developers often have to implement complex workarounds just to create simple functionalities. In Grug, iteration is natively supported thanks to our innovative approach of handling chain state.[15]
- **Null type**: EVM does not have the null type. That is, if a contract says a number is zero, it's impossible to tell whether this number is indeed zero, or that it's simply uninitiated. This ambiguity is a huge security pitfall. Grug supports null type (or rather, Rust supports null type and Grug just gets it for free).
- **Upgradability**: In Grug, contracts migration is supported natively by the VM. It's no longer necessary to rely on the complex and error-prune proxy pattern[16] as in EVM.
- **Readability**: Calldata in Grug are in JSON, a schema-less and human-readable format, while contract states are encoded in Borsh,[17] a compact and performant scheme.
- Reentrancy attack is made impossible[18] by design in Grug.

2 The choice of Rust and WebAssembly

We explicitly make the choice to NOT invent a new language, in contrary to what some of our competitors do (Solidity, Move, Fuel). Inventing a new language means you have to reinvent all the tooling, rewrite and re-audit all the libraries; not worth it.

Furthermore, your compiler probably won't be very good. A compiler is a tremendously complex piece of software; a domain-specific language, with its limited user base, is unlikely to have the resource to spend on perfecting it. This is why we often see Solidity devs resorting to writing assembly code - the compiler simply isn't capable of producing gas-efficient bytecodes.

Grug smart contracts are written in Rust, the most admired programming language 8 years in a row,[19] and compile to WebAssembly (Wasm).

While calling contracts, Grug uses Wasmer,[20] a highly optimized Wasm runtime that utilizes a just-in-time (JIT) compiler to achieve near native performance,[21] which can be 10-100x cheaper in computation and up to 100-500x cheaper in memory usage compared to single-threaded EVM, according to benchmarks[22] done by Arbitrum, which also uses Wasmer for its Stylus[23] framework.

Wasm is also supported by all modern web browsers. This opens up new possibilities for frontend develpement (check out this presentation[24]).

Another cool thing about using Rust/Wasm is that it often gets better without us having to do anything. For instance, the Rust 1.63 release reduces compiled contract sizes by up to 40%.[25] We enjoy the contributions from numerous Rust/Wasm users outside of the crypto space, a luxury that domain-specific languages don't have.

3 Sequencing

To create a fully functioning blockchain, Grug needs to be coupled with a **sequencing layer**. If you couple it with a layer-1 consensus protocol, you get an L1 chain; couple it with a layer-2 sequencer, you get an L2 chain.

Grug talks with the sequencing layer through the Application-Blockchain Interface (ABCI).[26] Any ABCI-compatible sequcing solution should work, such as:

- CometBFT[27]
- CometBLS[28]
- Block SDK[29]
- Rollchains[30]
- Dymint[31]
- ABCI wrapper of op-node[32] (WIP by Polymer team[33])

4 Future plans

Grug V1 won't come with **verifiable computation**; our team set this aside for now and focus on shipping some apps first. Eventually though, we plan to make Grug ZK-provable utilizing one of the zkVMs: RiscZero,[34] SP1,[35] Jolt,[36] Fluent,[37] or zkLLVM.[38]

Besides these, we will also investigate **state eviction**, which is a solution to state bloating, believed to be the greatest obstacle for Ethereum scaling.[39] Our solution will be similar to what is proposed in the Diem whitepaper (§4.4).[40]

We will also investigate **parallel transaction processing** using Block-STM.[41]

5 Acknowledgements

Grug is heavily inspired by CosmWasm,[42] both in architecture design and in borrowing actual code (in compliance with the Apache-2.0 license). It's safe to say that Grug simply won't exist without the pioneering work by CosmWasm. Our big thanks to the CosmWasm creators.

References

[1] Sam Wilson et al. *EIP-3074: AUTH and AUTHCALL opcodes*. Oct. 2020. URL: https://eips.ethereum.org/EIPS/eip-3074.

[2] Larry Engineer. *abstract-account*. 2023. URL: https://github.com/larry0x/abstract-account.

[3] Burnt. *Xion*. URL: https://x.com/burnt_xion.

[4] Larry Engineer. *A generalized credit protocol utilizing Mars lending market*. June 2022. URL: https://forum.marsprotocol.io/t/a-generalized-credit-protocol-utilizing-mars-lending-market/598.

[5] _gabrielShapir0. June 2022. URL: https://x.com/lex_node/status/1542214455517388803.

[6] Mars Protocol. URL: https://marsprotocol.io/.

[7] Gearbox Protocol. URL: https://gearbox.fi/.

[8] Osmosis. URL: https://docs.osmosis.zone/overview/features/protorev/.

[9] Cosmos SDK. URL: https://docs.cosmos.network/v0.50/build/building-modules/beginblock-endblock.

[10] Osmosis. URL: https://docs.osmosis.zone/osmosis-core/modules/tokenfactory/.

[11] Metaplex. URL: https://www.metaplex.com/.

[12] IBC Protocol. URL: https://www.ibcprotocol.dev/.
[13] Circle. URL: https://www.circle.com/en/cross-chain-transfer-protocol.
[14] noble. URL: https://nobleassets.xyz/.
[15] Cosmos SDK. URL: https://github.com/cosmos/cosmos-sdk/blob/main/docs/architecture/adr-065-store-v2.md.
[16] OpenZeppelin. URL: https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies.
[17] Near Protocol. URL: https://github.com/near/borsh.
[18] CosmWasm. URL: https://docs.cosmwasm.com/docs/architecture/smart-contracts#avoiding-reentrancy-attacks.
[19] StackOverflow. URL: https://survey.stackoverflow.co/2023/#section-admired-and-desired-programming-scripting-and-markup-languages.
[20] Wasmer. URL: https://wasmer.io/.
[21] Brandon Fish. *Benchmarking WebAssembly Runtimes*. URL: https://medium.com/wasmer/benchmarking-webassembly-runtimes-18497ce0d76e.
[22] Arbitrum. URL: https://docs.arbitrum.io/stylus/concepts/stylus-gas#stylus-gas-costs.
[23] Arbitrum. URL: https://arbitrum.io/stylus.
[24] Gabe Rodriguez. *WASM Cross-Platform*. July 2023. URL: https://www.youtube.com/live/xe1iQ-FHviY?t=5755.
[25] Simon Warta. Sept. 2022. URL: https://x.com/simon_warta/status/1565454352701169664.
[26] Tendermint. URL: https://docs.cometbft.com/v0.37/spec/abci/.
[27] CometBFT. URL: https://github.com/cometbft/cometbft.
[28] Union Labs. URL: https://docs.union.build/architecture/cometbls/.
[29] Skip Protocol. URL: https://github.com/skip-mev/block-sdk.
[30] Rollchains. URL: https://rollchains.com/.
[31] Dymension. URL: https://github.com/dymensionxyz/dymint.
[32] Optimism. URL: https://github.com/ethereum-optimism/optimism/tree/develop/op-node.
[33] Polymer Labs. URL: https://x.com/Polymer_Labs.
[34] RISC Zero. URL: https://x.com/RiscZero.
[35] Succinct Labs. URL: https://x.com/SuccinctLabs.
[36] a16z. URL: https://github.com/a16z/jolt.
[37] Fluent. URL: https://x.com/fluentxyz.
[38] =nil; Foundation. URL: https://x.com/nil_foundation.
[39] Péter Szilágyi. *Ethereum in numbers: Where TPS meets physics*. Sept. 2023. URL: https://www.youtube.com/watch?v=Cmuz_Xn_YJw.
[40] Diem Association. URL: https://developers.diem.com/docs/technical-papers/the-diem-blockchain-paper/.
[41] Rati Gelashvili et al. *Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing*. 2022. arXiv: 2203.06871 [cs.DC]. URL: https://arxiv.org/abs/2203.06871.
[42] CosmWasm. URL: https://cosmwasm.com/.