Overview of the Solana Runtime

Capability of Programs#

The runtime only permits the owner program to debit the account or modify its data. The program then defines additional rules for whether the client can modify accounts it owns. In the case of the System program, it allows users to transfer lamports by recognizing transaction signatures. If it sees the client signed the transaction using the keypair'sprivate key , it knows the client authorized the token transfer.

In other words, the entire set of accounts owned by a given program can be regarded as a key-value store, where a key is the account address and value is program-specific arbitrary binary data. A program author can decide how to manage the program's whole state, possibly as many accounts.

After the runtime executes each of the transaction's instructions, it uses the account metadata to verify that the access policy was not violated. If a program violates the policy, the runtime discards all account changes made by all instructions in the transaction, and marks the transaction as failed.

Policy#

After a program has processed an instruction, the runtime verifies that the program only performed operations it was permitted to, and that the results adhere to the runtime policy.

The policy is as follows:

- Only the owner of the account may change owner.* And only if the account is writable.
-
    - And only if the account is not executable.
-
    - And only if the data is zero-initialized or empty.
- An account not assigned to the program cannot have its balance decrease.
- The balance of read-only and executable accounts may not change.
- Only the owner may change account size and data.* And if the account is writable.
-
    - And if the account is not executable.
- Executable is one-way (false->true) and only the account owner may set it.
- No one can make modifications to the rent_epoch associated with this account.

Balancing the balances#

Before and after each instruction, the sum of all account balances must stay the same. E.g. if one account's balance is increased, another's must be decreased by the same amount. Because the runtime can not see changes to accounts which were not passed to it, all accounts for which the balances were modified must be passed, even if they are not needed in the called instruction.

Compute Budget#

To prevent abuse of computational resources, each transaction is allocated a compute budget. The budget specifies a maximum number of compute units that a transaction can consume, the costs associated with different types of operations the transaction may perform, and operational bounds the transaction must adhere to.

As the transaction is processed compute units are consumed by its instruction's programs performing operations such as executing SBF instructions, calling syscalls, etc... When the transaction consumes its entire budget, or exceeds a bound such as attempting a call stack that is too deep, or loaded account data size exceeds limit, the runtime halts the transaction processing and returns an error.

The following operations incur a compute cost:

- Executing SBF instructions
- Passing data between programs
- Calling system calls* logging
-
    - creating program addresses
-
    - cross-program invocations
-
    - ...

For cross-program invocations, the instructions invoked inherit the budget of their parent. If an invoked instruction consumes the transactions remaining budget, or exceeds a bound, the entire invocation chain and the top level transaction processing are halted.

The currentcompute budget can be found in the Solana Program Runtime.

Example Compute Budget#

For example, if the compute budget set in the Solana runtime is:

max_units: 1,400,000, log_u64_units: 100, create_program address units: 1500, invoke_units: 1000, max_invoke_stack_height: 5, max_instruction_trace_length: 64, max_call_depth: 64, stack_frame_size: 4096, log_pubkey_units: 100, ... Then any transaction:

- Could execute 1,400,000 SBF instructions, if it did nothing else.
- Cannot exceed 4k of stack usage.
- Cannot exceed a SBF call depth of 64.
- Cannot exceed invoke stack height of 5 (4 levels of cross-program
- invocations).

NOTE: Since the compute budget is consumed incrementally as the transaction executes, the total budget consumption will be a combination of the various costs of the operations it performs. At runtime a program may log how much of the compute budget remains. Seedebugging for more information.

Prioritization fees#

As part of the Compute Budget, the runtime supports transactions including anoptional fee to prioritize itself against others known as aprioritization fee .

Thisprioritization fee is calculated by multiplying the number ofcompute units by thecompute unit price (measured in micro-lamports). These values may be set via the Compute Budget instructionsSetComputeUnitLimit andSetComputeUnitPrice once per transaction.

You can learn more of the specifics ofhow andwhen to set a prioritization fee on thetransaction fees page.

Accounts data size limit#

A transaction should request the maximum bytes of accounts data it is allowed to load by including aSetLoadedAccountsDataSizeLimit instruction, requested limit is capped byMAX_LOADED_ACCOUNTS_DATA_SIZE_BYTES . If noSetLoadedAccountsDataSizeLimit is provided, the transaction is defaulted to have limit ofMAX_LOADED_ACCOUNTS_DATA_SIZE_BYTES .

TheComputeBudgetInstruction::set_loaded_accounts_data_size_limit function can be used to create this instruction:

let instruction = ComputeBudgetInstruction::set_loaded_accounts_data_size_limit(100_000);

New Features#

As Solana evolves, new features or patches may be introduced that changes the behavior of the cluster and how programs run. Changes in behavior must be coordinated between the various nodes of the cluster. If nodes do not coordinate, then these changes can result in a break-down of consensus. Solana supports a mechanism called runtime features to facilitate the smooth adoption of changes.

Runtime features are epoch coordinated events where one or more behavior changes to the cluster will occur. New changes to Solana that will change behavior are wrapped with feature gates and disabled by default. The Solana tools are then used to activate a feature, which marks it pending, once marked pending the feature will be activated at the next epoch.

To determine which features are activated use theSolana command-line tools :

solana feature status If you encounter problems, first ensure that the Solana tools version you are using match the version returned bysolana cluster-version . If they do not match,install the correct tool suite .