Contract Optimization on OP Mainnet

OP Mainnet is designed to be EVM equivalent(opens in a new tab) , which means OP Mainnet looks exactly like Ethereum in every way possible. One large and mostly unavoidable discrepancy between OP Mainnet and Ethereum is a slight difference in transaction fee models . This difference means that there are some opportunities to optimize OP Mainnet contracts to better take advantage of the OP Mainnet fee model.

This guide explains the basic concepts that allow you to optimize OP Mainnet contracts and provides several examples of how you might take advantage of these concepts. You'll also get a better understanding of the tradeoffs and risks involved with potential contract optimizations. By the end of this guide you should have a clear understanding of how OP Mainnet contracts can be optimized and whether or not these optimizations make sense for your application.

Contract optimizations are powerful but can also create additional contract complexity. Some types of optimizations may also become unnecessary as OP Mainnet evolves. Make sure to read through all of the considerations on this page before committing to any optimizations.

Fundamentals

App developers might already be familiar with the concept of gas optimization(opens in a new tab) to decrease the cost of interacting with smart contracts. Gas optimization can reduce the amount of gas used by a contract and make your application cheaper (and therefore improve user experience).

OP Mainnet contracts can be gas optimized to reduce overall transaction costs, just like contracts on Ethereum. However, OP Mainnet transactions must also pay for another fee called the L1 Data Fee which accounts for the cost of publishing transaction data to Ethereum. At the time this guide was written, this L1 Data Fee makes up the majority of the cost of most transactions on OP Mainnet.

Because the L1 Data Fee tends to be larger than the execution gas fee,OP Mainnet contracts can be further optimized by increasing the amount of storage/execution used in order to decrease the amount of user-provided data in each transaction. This is the basic premise that makes OP Mainnet contract optimization slightly different than on Ethereum.

Considerations

Additional Complexity

Contract optimizations can create additional complexity, which can in turn create additional risk. Developers should always consider whether this added complexity is worth the reduction in cost.

Changing Economics

Various potential upgrades to OP Mainnet may also make optimizations to the L1 Data Fee less relevant. For instance EIP-4844(opens in a new tab) , if adopted, should significantly reduce the cost of publishing data to Ethereum and would therefore also decrease the L1 Data Fee.

OP Mainnet also uses an EIP-1559(opens in a new tab) mechanism that automatically increases the base fee as chain usage increases. Optimizations based on reducing the L1 Data Fee may become less relevant if the base fee becomes a larger part of the total transaction cost.

Generally speaking,developers should assume that L1 Data Fee optimizations will become less useful as time goes on . If you expect your contracts to be used for long periods of time, you may wish to avoid optimizations that can potentially decrease long-term gas efficiency. If you expect your contracts to be used mostly in the short term or you have the ability to upgrade contracts in the future, optimizations may be better suited for you.

Techniques

Calldata Compression

Compressing user data on the client side and decompressing it on the contract side can be an effective way to decrease costs on OP Mainnet. This technique decreases the amount of data provided by the user in exchange for a significant increase in onchain computation.

Although several libraries exist to perform this calldata compression, none of these libraries have been audited as of the writing of this page. As a result, links to these libraries have been explicitly omitted here to avoid encouraging developers from using potentially buggy software. Most of these libraries can be found with a quick search online but, as always, be careful with code you find on the internet.

Custom Encoding

TheSolidity ABI encoding scheme(opens in a new tab) is not particularly data efficient. Contracts can often reduce the amount of calldata provided by defining a custom argument encoding protocol.

Custom argument encodings can be combined with stored data to further reduce costs. For example,address arguments could potentially be replaced with auint64 pointer that performs a lookup in a stored mapping ofuint64 => address . This would cut out a significant number of bytes with further reductions if the total number of addresses that need to be looked up is small (which would allowuint64 to be reduced touint32 or less).

Custom encodings are typically less complex that general-purpose calldata compression libraries but carry additional complexity no matter what. When combined with stored data, application-specific custom encodings can be significantly more efficient than general-purpose compression mechanisms.

System Contracts Transaction Fees