Object-Based Local Fee Markets

Object-based local fee markets limit the rate of transactions writing to a single shared object, preventing the network from becoming overloaded with checkpoints that take too long to execute.

The Sui network's object-based architecture allows processing many different user transactions massively in parallel, in a way that's not possible on most other networks. However, if multiple transactions are all writing to the same shared object, they must execute in sequential order. There is a limit to how many transactions the network can process that touch one specific object.

If you see transactions fail with the error ExecutionCancelledDueToSharedObjectCongestion , you are observing object-based local fee markets at work. Continue reading to learn:

Sui's local fee market algorithm runs every time a new batch of sequenced transactions is received from consensus.

Sorts all transactions in order of gas price, from highest to lowest.

Estimates each transaction's execution cost using the TotalGasBudgetWithCap heuristic.

The SharedObjectCongestionTracker keeps a running tally of how much per-object congestion budget is used. If all shared objects used by the transaction have enough budget left, the transaction is scheduled. In that case, it consumes budget for the mutable shared objects that it uses. Keep in mind that immutable shared objects don't consume any budget, because multiple immutable uses of a shared object can execute in parallel. If the transaction cannot be scheduled, it is deferred until the next commit.

Transaction priority is determined solely by gas price. If you want your transaction to be assigned access to shared objects ahead of others in the same consensus commit, you must pay a higher gas price than the others. This is the only way to get priority access.

Using a gas price that is at least five times the reference gas price also increases the likelihood that the transaction is included in the earliest consensus commit possible. See SIP-45 for details.

An improved algorithm is in active development for estimating transaction execution cost. This topic will be updated to reflect any future changes.

Sui limits the per-commit execution capacity of each shared object. If transactions touching a shared object have a low estimated cost, more of them can fit. If they have a high estimated cost, not as many can fit.

The cost of a transaction for local fee market purposes is estimated using the TotalGasBudgetWithCap heuristic.

A transaction's initial estimated cost is simply its gas budget. (Unfortunately, the final gas cost of a transaction is not known until after execution, which is too late to use in scheduling.)

For transactions with a very high gas budget, the estimated cost is then lowered to an upper limit that is determined based on the number of MoveCall s and the number of Move Input s in each call.

If the network is unable to schedule a transaction because it's trying to use a congested shared object with no space left, the transaction is deferred. This means that validators hold onto the transaction and attempt to schedule it with the next commit, each time prioritizing all waiting transactions by gas price.

If a transaction is deferred for several commits without successfully being scheduled, it's cancelled and returns an ExecutionCancelledDueToSharedObjectCongestion error. You'll need to try it again with a higher gas price, or at a time when the shared objects it depends on are not as congested.

At the protocol level, Sui is configured with a fixed per-commit limit and burst capacity for each shared object. On average, a shared object's activity cannot exceed the per-commit limit. However, short bursts of traffic might exceed the limit temporarily.

There is no way to increase the total execution capacity of a particular shared object. The network sets that limit to ensure that validators, Full nodes, and indexers can all execute checkpoints in a reasonable amount of time.

For priority access to a congested shared object, set a higher gas price.

Set a gas budget that is close to your actual execution cost. You can use dry runs to estimate this. Be aware that some transactions have variable cost.

When designing Move packages, avoid using a single shared object if possible. For example, a DEX application with a single, main shared object and dynamic fields for each currency pair would suffer much more congestion than one with a separate object per currency pair.

If your object O is congested, optimize the gas usage of the functions that are commonly used to access the object. For example, if all the accesses to your object happen via Move function f , halving the gas usage of f will effectively double the number of transactions that can touch O in a single commit (assuming the gas budget is correspondingly decreased).

How object-based local fee markets work

Sui's local fee market algorithm runs every time a new batch of sequenced transactions is received from consensus.

Sorts all transactions in order of gas price, from highest to lowest.

Estimates each transaction's execution cost using the TotalGasBudgetWithCap heuristic.

The SharedObjectCongestionTracker keeps a running tally of how much per-object congestion budget is used. If all shared objects used by the transaction have enough budget left, the transaction is scheduled. In that case, it consumes budget for the mutable shared objects that it uses. Keep in mind that immutable shared objects don't consume any budget, because multiple immutable uses of a shared object can execute in parallel. If the transaction cannot be scheduled, it is deferred until the next commit.

Transaction priority is determined solely by gas price. If you want your transaction to be assigned access to shared objects ahead of others in the same consensus commit, you must pay a higher gas price than the others. This is the only way to get priority access.

Using a gas price that is at least five times the reference gas price also increases the likelihood that the transaction is included in the earliest consensus commit possible. See SIP-45 for details.

An improved algorithm is in active development for estimating transaction execution cost. This topic will be updated to reflect any future changes.

Sui limits the per-commit execution capacity of each shared object. If transactions touching a shared object have a low estimated cost, more of them can fit. If they have a high estimated cost, not as many can fit.

The cost of a transaction for local fee market purposes is estimated using the TotalGasBudgetWithCap heuristic.

A transaction's initial estimated cost is simply its gas budget. (Unfortunately, the final gas cost of a transaction is not known until after execution, which is too late to use in scheduling.)

For transactions with a very high gas budget, the estimated cost is then lowered to an upper limit that is determined based on the number of MoveCall s and the number of Move Input s in each call.

If the network is unable to schedule a transaction because it's trying to use a congested shared object with no space left, the transaction is deferred. This means that validators hold onto the transaction and attempt to schedule it with the next commit, each time prioritizing all waiting transactions by gas price.

If a transaction is deferred for several commits without successfully being scheduled, it's cancelled and returns an ExecutionCancelledDueToSharedObjectCongestion error. You'll need to try it again with a higher gas price, or at a time when the shared objects it depends on are not as congested.

At the protocol level, Sui is configured with a fixed per-commit limit and burst capacity for each shared object. On average, a shared object's activity cannot exceed the per-commit limit. However, short bursts of traffic might exceed the limit temporarily.

There is no way to increase the total execution capacity of a particular shared object. The network sets that limit to ensure that validators, Full nodes, and indexers can all execute checkpoints in a reasonable amount of time.

For priority access to a congested shared object, set a higher gas price.

Set a gas budget that is close to your actual execution cost. You can use dry runs to estimate this. Be aware that some transactions have variable cost.

When designing Move packages, avoid using a single shared object if possible. For example, a DEX application with a single, main shared object and dynamic fields for each currency pair would suffer much more congestion than one with a separate object per currency pair.

If your object O is congested, optimize the gas usage of the functions that are commonly used to access the object. For example, if all the accesses to your object happen via Move function f , halving the gas usage of f will effectively double the number of

transactions that can touch O in a single commit (assuming the gas budget is correspondingly decreased).

Practical takeaways

For priority access to a congested shared object, set a higher gas price.

Set a gas budget that is close to your actual execution cost. You can use dry runs to estimate this. Be aware that some transactions have variable cost.

When designing Move packages, avoid using a single shared object if possible. For example, a DEX application with a single, main shared object and dynamic fields for each currency pair would suffer much more congestion than one with a separate object per currency pair.

If your object O is congested, optimize the gas usage of the functions that are commonly used to access the object. For example, if all the accesses to your object happen via Move function f, halving the gas usage of f will effectively double the number of transactions that can touch O in a single commit (assuming the gas budget is correspondingly decreased).