Validator Committee

A set of independent validators participate on the Sui network, each running its own instance of the Sui software on a separate machine (or a sharded cluster of machines the same entity operates). Each validator handles read and write requests sent by clients, verifying transactions and updating on-chain information.

To learn how to set up and run a Sui Validator node, including how staking and rewards work, see Sui Validator Node Configuration .

Sui uses Delegated Proof-of-Stake (DPoS) to determine which validators operate the network and their voting power. Validators are incentivized to participate in good faith via a share of transaction fees, staking rewards, and slashing stake and staking rewards in case of misbehavior.

Operation of the Sui network is temporally partitioned into non-overlapping, approximate fixed-duration (~24-hour) epochs. During a particular epoch, the set of validators participating in the network and their voting power is fixed. At an epoch boundary, reconfiguration might occur and can change the set of validators participating in the network and their voting power. Conceptually, reconfiguration starts a new instance of the Sui protocol with the previous epoch's final state as genesis and the new set of validators as the operators. Besides validator set changes, tokenomics operations such as staking/un-staking, and distribution of staking rewards are also processed at epoch boundaries.

A quorum is a set of validators whose combined voting power is greater than two-thirds (>2/3) of the total during a particular epoch. For example, in a Sui instance operated by four validators that all have the same voting power, any group containing three validators is a quorum.

The quorum size of >2/3 ensures Byzantine fault tolerance (BFT). A validator commits a transaction (durably store the transaction and update its internal state with the effects of the transaction) only if it is accompanied by cryptographic signatures from a quorum. Sui calls the combination of the transaction and the quorum signatures on its bytes a certificate . The policy of committing only certificates ensures Byzantine fault tolerance: if >2/3 of the validators faithfully follow the protocol, they are guaranteed to eventually agree on both the set of committed certificates and their effects.

A validator can handle two types of write requests: transactions and certificates. At a high level, a client:

When a validator receives a transaction from a client, it first performs transaction validity checks (validity of the sender's signature). If the checks pass, the validator locks all owned-objects and signs the transaction bytes, then returns the signature to the client. The client repeats this process with multiple validators until it has collected signatures on its transaction from a quorum, thereby forming a certificate.

The process of collecting validator signatures on a transaction into a certificate and the process of submitting certificates can be performed in parallel. The client can simultaneously multicast transactions/certificates to an arbitrary number of validators. Alternatively, a client can outsource either or both of these tasks to a third-party service provider. This provider must be trusted for liveness (it can refuse to form a certificate), but not for safety (it cannot change the effects of the transaction, and does not need the user's secret key).

After the client forms a certificate, it submits it to the validators, which perform certificate validity checks. These checks ensure the signers are validators in the current epoch, and the signatures are cryptographically valid. If the checks pass, the validators execute the transaction inside the certificate. Execution of a transaction either succeeds and commits all of its effects or aborts and has no effect other than debiting the transaction's gas input. Some reasons a transaction might abort include an explicit abort instruction, a runtime error such as division by zero, or exceeding the maximum gas budget. Whether it succeeds or aborts, the validator durably stores the certificate indexed by the hash of its inner transaction.

If a client collects a quorum of signatures on the effects of the transaction, then the client has a promise of finality. This means that transaction effects persist on the shared database and are actually committed and visible to everyone by the end of the epoch. This does not mean that the latency is a full epoch, because you can use the effects certificate to convince anyone of the transactions finality, as well as to access the effects and issue new transactions. As with transactions, you can parallelize the process of sharing a certificate with validators and (if desired) outsource to a third-party service provider.

Sui uses a high-throughput DAG-based consensus engine implementing the Mysticeti algorithm to sequence certified transactions. Consensus produces a series of commits. In each commit, there is a sequence of certified transactions.

The order of transactions in consensus output determines the relative order they can operate on each shared object. Note that executions of transactions touching different shared objects are parallelized on multiple cores.

If total execution cost of transactions in a consensus commit is over a threshold, transactions can be cancelled post consensus to avoid overloading the system.

Epochs

Operation of the Sui network is temporally partitioned into non-overlapping, approximate fixed-duration (~24-hour) epochs. During a particular epoch, the set of validators participating in the network and their voting power is fixed. At an epoch boundary, reconfiguration might occur and can change the set of validators participating in the network and their voting power. Conceptually, reconfiguration starts a new instance of the Sui protocol with the previous epoch's final state as genesis and the new set of validators as the operators. Besides validator set changes, tokenomics operations such as staking/un-staking, and distribution of staking rewards are also processed at epoch boundaries.

A quorum is a set of validators whose combined voting power is greater than two-thirds (>2/3) of the total during a particular epoch. For example, in a Sui instance operated by four validators that all have the same voting power, any group containing three validators is a quorum.

The quorum size of >2/3 ensures Byzantine fault tolerance (BFT). A validator commits a transaction (durably store the transaction and update its internal state with the effects of the transaction) only if it is accompanied by cryptographic signatures from a quorum. Sui calls the combination of the transaction and the quorum signatures on its bytes a certificate . The policy of committing only certificates ensures Byzantine fault tolerance: if >2/3 of the validators faithfully follow the protocol, they are guaranteed to eventually agree on both the set of committed certificates and their effects.

A validator can handle two types of write requests: transactions and certificates. At a high level, a client:

When a validator receives a transaction from a client, it first performs transaction validity checks (validity of the sender's signature). If the checks pass, the validator locks all owned-objects and signs the transaction bytes, then returns the signature to the client. The client repeats this process with multiple validators until it has collected signatures on its transaction from a quorum, thereby forming a certificate.

The process of collecting validator signatures on a transaction into a certificate and the process of submitting certificates can be performed in parallel. The client can simultaneously multicast transactions/certificates to an arbitrary number of validators. Alternatively, a client can outsource either or both of these tasks to a third-party service provider. This provider must be trusted for liveness (it can refuse to form a certificate), but not for safety (it cannot change the effects of the transaction, and does not need the user's secret key).

After the client forms a certificate, it submits it to the validators, which perform certificate validity checks. These checks ensure the signers are validators in the current epoch, and the signatures are cryptographically valid. If the checks pass, the validators execute the transaction inside the certificate. Execution of a transaction either succeeds and commits all of its effects or aborts and has no effect other than debiting the transaction's gas input. Some reasons a transaction might abort include an explicit abort instruction, a runtime error such as division by zero, or exceeding the maximum gas budget. Whether it succeeds or aborts, the validator durably stores the certificate indexed by the hash of its inner transaction.

If a client collects a quorum of signatures on the effects of the transaction, then the client has a promise of finality. This means that transaction effects persist on the shared database and are actually committed and visible to everyone by the end of the epoch. This does not mean that the latency is a full epoch, because you can use the effects certificate to convince anyone of the transactions finality, as well as to access the effects and issue new transactions. As with transactions, you can parallelize the process of sharing a certificate with validators and (if desired) outsource to a third-party service provider.

Sui uses a high-throughput DAG-based consensus engine implementing the Mysticeti algorithm to sequence certified transactions. Consensus produces a series of commits. In each commit, there is a sequence of certified transactions.

The order of transactions in consensus output determines the relative order they can operate on each shared object. Note that executions of transactions touching different shared objects are parallelized on multiple cores.

If total execution cost of transactions in a consensus commit is over a threshold, transactions can be cancelled post consensus to avoid overloading the system.

Quorums

A quorum is a set of validators whose combined voting power is greater than two-thirds (>2/3) of the total during a particular epoch. For example, in a Sui instance operated by four validators that all have the same voting power, any group containing three validators is a quorum.

The quorum size of >2/3 ensures Byzantine fault tolerance (BFT). A validator commits a transaction (durably store the transaction and update its internal state with the effects of the transaction) only if it is accompanied by cryptographic signatures from a quorum. Sui calls the combination of the transaction and the quorum signatures on its bytes a certificate . The policy of committing only certificates ensures Byzantine fault tolerance: if >2/3 of the validators faithfully follow the protocol, they are guaranteed to eventually agree on both the set of committed certificates and their effects.

A validator can handle two types of write requests: transactions and certificates. At a high level, a client:

When a validator receives a transaction from a client, it first performs transaction validity checks (validity of the sender's signature). If the checks pass, the validator locks all owned-objects and signs the transaction bytes, then returns the signature to the client. The client repeats this process with multiple validators until it has collected signatures on its transaction from a quorum, thereby forming a certificate.

The process of collecting validator signatures on a transaction into a certificate and the process of submitting certificates can be performed in parallel. The client can simultaneously multicast transactions/certificates to an arbitrary number of validators. Alternatively, a client can outsource either or both of these tasks to a third-party service provider. This provider must be trusted for liveness (it can refuse to form a certificate), but not for safety (it cannot change the effects of the transaction, and does not need the user's secret key).

After the client forms a certificate, it submits it to the validators, which perform certificate validity checks. These checks ensure the signers are validators in the current epoch, and the signatures are cryptographically valid. If the checks pass, the validators execute the transaction inside the certificate. Execution of a transaction either succeeds and commits all of its effects or aborts and has no effect other than debiting the transaction's gas input. Some reasons a transaction might abort include an explicit abort instruction, a runtime error such as division by zero, or exceeding the maximum gas budget. Whether it succeeds or aborts, the validator durably stores the certificate indexed by the hash of its inner transaction.

If a client collects a quorum of signatures on the effects of the transaction, then the client has a promise of finality. This means that transaction effects persist on the shared database and are actually committed and visible to everyone by the end of the epoch. This does not mean that the latency is a full epoch, because you can use the effects certificate to convince anyone of the transactions finality, as well as to access the effects and issue new transactions. As with transactions, you can parallelize the process of sharing a certificate with validators and (if desired) outsource to a third-party service provider.

Sui uses a high-throughput DAG-based consensus engine implementing the Mysticeti algorithm to sequence certified transactions. Consensus produces a series of commits. In each commit, there is a sequence of certified transactions.

The order of transactions in consensus output determines the relative order they can operate on each shared object. Note that executions of transactions touching different shared objects are parallelized on multiple cores.

If total execution cost of transactions in a consensus commit is over a threshold, transactions can be cancelled post consensus to avoid overloading the system.

Write requests

A validator can handle two types of write requests: transactions and certificates. At a high level, a client:

When a validator receives a transaction from a client, it first performs transaction validity checks (validity of the sender's signature). If the checks pass, the validator locks all owned-objects and signs the transaction bytes, then returns the signature to the client. The client repeats this process with multiple validators until it has collected signatures on its transaction from a quorum, thereby forming a certificate.

The process of collecting validator signatures on a transaction into a certificate and the process of submitting certificates can be performed in parallel. The client can simultaneously multicast transactions/certificates to an arbitrary number of validators. Alternatively, a client can outsource either or both of these tasks to a third-party service provider. This provider must be trusted for liveness (it can refuse to form a certificate), but not for safety (it cannot change the effects of the transaction, and does not need the user's secret key).

After the client forms a certificate, it submits it to the validators, which perform certificate validity checks. These checks ensure the signers are validators in the current epoch, and the signatures are cryptographically valid. If the checks pass, the validators execute the transaction inside the certificate. Execution of a transaction either succeeds and commits all of its effects or aborts and has no effect other than debiting the transaction's gas input. Some reasons a transaction might abort include an explicit abort instruction, a runtime error such as division by zero, or exceeding the maximum gas budget. Whether it succeeds or aborts, the validator durably stores the certificate indexed by the hash of its inner transaction.

If a client collects a quorum of signatures on the effects of the transaction, then the client has a promise of finality. This means that transaction effects persist on the shared database and are actually committed and visible to everyone by the end of the epoch. This does not mean that the latency is a full epoch, because you can use the effects certificate to convince anyone of the transactions finality, as well as to access the effects and issue new transactions. As with transactions, you can parallelize the process of sharing a certificate with validators and (if desired) outsource to a third-party service provider.

Sui uses a high-throughput DAG-based consensus engine implementing the Mysticeti algorithm to sequence certified transactions. Consensus produces a series of commits. In each commit, there is a sequence of certified transactions.

The order of transactions in consensus output determines the relative order they can operate on each shared object. Note that executions of transactions touching different shared objects are parallelized on multiple cores.

If total execution cost of transactions in a consensus commit is over a threshold, transactions can be cancelled post consensus to avoid overloading the system.

The role of consensus

Sui uses a high-throughput DAG-based consensus engine implementing the Mysticeti algorithm to sequence certified transactions. Consensus produces a series of commits. In each commit, there is a sequence of certified transactions.

The order of transactions in consensus output determines the relative order they can operate on each shared object. Note that executions of transactions touching different shared objects are parallelized on multiple cores.

If total execution cost of transactions in a consensus commit is over a threshold, transactions can be cancelled post consensus to avoid overloading the system.