assumeutxo

Assumeutxo is a feature that allows fast bootstrapping of a validating bitcoind instance with a very similar security model to assumevalid.

The RPC commands `dumptxoutset` and `loadtxoutset` are used to respectively generate and load UTXO snapshots. The utility script `./contrib/devtools/utxo_snapshot.sh` may be of use.

General background

- assumeutxo proposal
- Github issue
- draft PR

Design notes

- A new block index `nStatus` flag is introduced, `BLOCK_ASSUMED_VALID`, to mark block index entries that are required to be assumed-valid by a chainstate created from a UTXO snapshot. This flag is mostly used as a way to modify certain CheckBlockIndex() logic to account for index entries that are pending validation by a chainstate running asynchronously in the background. We also use this flag to control which index entries are added to setBlockIndexCandidates during LoadBlockIndex().
- Indexing implementations via BaseIndex can no longer assume that indexation happens sequentially, since background validation chainstates can submit BlockConnected events out of order with the active chain.
- The concept of UTXO snapshots is treated as an implementation detail that lives behind the ChainstateManager interface. The external presentation of the changes required to facilitate the use of UTXO snapshots is the understanding that there are now certain regions of the chain that can be temporarily assumed to be valid (using the nStatus flag mentioned above). In certain cases, e.g. wallet rescanning, this is very similar to dealing with a pruned chain.

  Logic outside ChainstateManager should try not to know about snapshots, instead preferring to work in terms of more general states like assumed-valid.

Chainstate phases

Chainstate within the system goes through a number of phases when UTXO snapshots are used, as managed by `ChainstateManager`. At various points there can be multiple `CChainState` objects in existence to facilitate both maintaining the network tip and performing historical validation of the assumed-valid chain.

It is worth noting that though there are multiple separate chainstates, those chainstates share use of a common block index (i.e. they hold the same `BlockManager` reference).

The subheadings below outline the phases and the corresponding changes to chainstate data.

"Normal" operation via initial block download

`ChainstateManager` manages a single CChainState object, for which `m_snapshot_blockhash` is null. This chainstate is (maybe obviously) considered active. This is the "traditional" mode of operation for bitcoind.

number of chainstates	1
active chainstate	ibd

User loads a UTXO snapshot via `loadtxoutset` RPC

`ChainstateManager` initializes a new chainstate (see `ActivateSnapshot()`) to load the snapshot contents into. During snapshot load and validation (see `PopulateAndValidateSnapshot()`), the new chainstate is not considered active and the original chainstate remains in use as active.

number of chainstates	2
active chainstate	ibd

Once the snapshot chainstate is loaded and validated, it is promoted to active chainstate and a sync to tip begins. A new chainstate directory is created in the datadir for the snapshot chainstate called `chainstate_[SHA256 blockhash of snapshot base block]`.

number of chainstates	2
active chainstate	snapshot

The snapshot begins to sync to tip from its base block, technically in parallel with the original chainstate, but it is given priority during block download and is allocated most of the cache (see `MaybeRebalanceCaches()` and usages) as our chief consideration is getting to network tip.

Failure consideration: if shutdown happens at any point during this phase, both chainstates will be detected during the next init and the process will resume.

Snapshot chainstate hits network tip

Once the snapshot chainstate leaves IBD, caches are rebalanced (via `MaybeRebalanceCaches()` in `ActivateBestChain()`) and more cache is given to the background chainstate, which is responsible for doing full validation of the assumed-valid parts of the chain.

Note: at this point, ValidationInterface callbacks will be coming in from both chainstates. Considerations here must be made for indexing, which may no longer be happening sequentially.

Background chainstate hits snapshot base block

Once the tip of the background chainstate hits the base block of the snapshot chainstate, we stop use of the background chainstate by setting `m_stop_use` (not yet committed - see #15606), in `CompleteSnapshotValidation()`, which is checked in `ActivateBestChain()`). We hash the background chainstate's UTXO set contents and ensure it matches the compiled value in `CMainParams::m_assumeutxo_data`.

The background chainstate data lingers on disk until shutdown, when in `ChainstateManager::Reset()`, the background chainstate is cleaned up with `ValidatedSnapshotShutdownCleanup()`, which renames the `chainstate_[hash]` datadir as `chainstate`.

number of chainstates	2 (ibd has `m_stop_use=true`)
active chainstate	snapshot

Failure consideration: if bitcoind unexpectedly halts after `m_stop_use` is set on the background chainstate but before `CompleteSnapshotValidation()` can finish, the need to complete snapshot validation will be detected on subsequent init by `ChainstateManager::CheckForUncleanShutdown()`.

Bitcoind restarts sometime after snapshot validation has completed

When bitcoind initializes again, what began as the snapshot chainstate is now indistinguishable from a chainstate that has been built from the traditional IBD process, and will be initialized as such.

number of chainstates	1
active chainstate	ibd