chaincode

How Bitcoin Core works

LoadMempool() and DumpMempool()

Overview

We want

- Transactions
- (Our own) priority for these transactions

to be persistent across restarts, crashes and power outages.

These two functions do that for us:

- **`DumpMempool()`** writes mempool and transaction priority to **`mempool.dat`**
- **`LoadMempool()`** reads mempool and transaction priority from **`mempool.dat`**

The mempool.dat file is located in the bitcoin `-datadir`.

Manually prioritizing transactions

```
bitcoin-cli getmempoolentry 5783dd3f...
{
  "fees": {
    "base": 0.00080000,
    "modified": 0.00080000
    "ancestor": 0.000
    "descenda
  },
    : 0.00080000,
  "modifiedfee": 0.00080000,
  ...
}
```

```
bitcoin-cli prioritisetransaction 5783dd3f... 0 10000
```

```
bitcoin-cli getmempoolentry 5783dd3f...
{
  "fees": {
    "base": 0.00080000,
    "modified": 0.00090000,
    "ancestor": 0.00090000,
    "descendant": 0.00090000
  },
  "vsize": 707,
  "fee": 0.00080000,
  "modifiedfee": 0.00090000,
  …
}
```

mempool.dat file structure

```
struct File{
  Header header;                    // information about the file
  vector<Entry> entries;            // mempool entries
  map<uint256, CAmount> mapDeltas;  // tx prioritization
}
```

```
struct Header {
 uint64_t version;                  // file version
 uint64_t entryCount;               // transaction count in file
};
```

```
struct Entry  {
  CTransaction tx;                  // raw tx
  int64_t nTime;                    // mempool entry time
  int64_t nFeeDelta;                // tx prioritization
};
```

chaincode

LoadMempool() in validation.cpp

`LoadMempool()` is called in the import thread (after loading/reindexing/activation of blocks)

1. Open the `mempool.dat` to read from
2. Read the file `header` (**`version`** and **`entryCount`**)
3. Check **`header.version == MEMPOOL_DUMP_VERSION`** `// currently 1`
4. Read entries (for the number of entries specified by **`header.entryCount`**)
   a. Prioritize the transaction (if prioritization data exist)
   b. (Try to) accept the transaction to the mempool
   c. Count **`valid`**, **`invalid`**, **`expired`** and **`already-there`** transactions
5. Read extra transaction prioritization data from `mapDeltas`
6. Print transaction import statistics

```
Imported mempool transactions from disk: 463 succeeded, 10 failed, 0 expired, 0 already there.
```

Current behavior of mempool persisting

Persisting to `mempool.dat` added in PR #8448 and first included in Bitcoin Core v0.14.0

- `LoadMempool()` is executed after after loading/reindexing/activation of blocks
  - In background (`getmempoolinfo` returns `"loaded": true` when loaded)
  - No real problem if file is not loadable
  - Log message: statistics about the validity of the processed transactions
- `DumpMempool()` is executed at shutdown
  - No real problem if file is not writeable
  - Log message: time taken to dump the transactions

Can be manually disabled with **`-persistmempool=0`** introduced in PR #9966 (v0.15.0)

Manually save to `mempool.dat` with the **`savemempool`** RPC introduced in PR #11099 (v.0.16.0)

Thank you and questions?

My debug.log at startup

Not a transaction

```
PrioritiseTransaction:
0000000000000000001ef7e2989941f6fba4a9bbae418de42e26e413b36a53ab feerate
+= 0.19999998

Imported mempool transactions from disk: 2199 succeeded, 0 failed, 0
expired, 0 already there
```