RSK Merge Mining (RMM) and Bitcoin Interoperability

Rev: 1.2

Date: June 26th, 2016

By Sergio Demian Lerner, RSK Chief Scientist

Introduction

Bitcoin mining is currently 98% based on mining-pools. Mining pools are organized as server (or a server farm) and many clients or miners. The server runs a Ming Pool Server Software (poolserver) such as:

- ckpool - Open source pool/database/proxy/passthrough/library in c for Linux
- Eloipool - Fast Python3 poolserver
- CoiniumServ - High performance C# Mono/.Net poolserver.
- Remote miner - mining pool software
- node-stratum-pool – High performance Stratum poolserver in Node.js
- ecoinpool - Erlang poolserver (not maintained)
- Pushpoold - Old mining poolserver in C (not maintained)
- Poold - Old Python mining poolserver (not maintained)
- PoolServerJ - Java mining poolserver (not maintained)

The poolserver generally communicates with a bitcoind instance though an RPC channel. The channel is used for obtaining which is the best branch tip to mine on top of (the parent block hash), and also to obtain selected transactions from the transaction pool to choose from in order to build a new block. This is done by the RPC command getblocktemplate. Also poolserver software generally polls bitcoind every second to detect a change in the parent block. To obtain lower latency on detecting new blocks, some poolserver softwares also connect to bitcoind via the p2p interface or use the blocknotify option to be notified. The Poolserver software can make use of one or more standard p2p connections to the Bitcoin network in order to submit the newly created blocks.

Merge-Mining

Merge mining is a technique for using the same mining power to secure a secondary blockchain. This is done by embedding a hash of the block of the secondary blockchain (a tag) in the block of the primary blockchain that is being mined. For the RSK case, the first blockchain is Bitcoin, and the second is RSK. The RSK blockchain interprets the PoW of a Bitcoin header and finds the tag that uniquely establishes a relation to a RSK header, and therefore associates Bitcoin PoW to a RSK header. The difficulty for the RSK blockchain will generally be lower than the difficulty of the Bitcoin blockchain, so many Bitcoin headers that are not a solution to the Bitcoin PoW puzzle will be valid solutions to the RSK PoW puzzle. Internally, each blockchain has a “target”. The

target is a 256-bit number that establishes the difficulty. The block header hash must be lower than the target for the block to be a PoW puzzle solution for the blockchain. The RSK target will be generally much higher than the Bitcoin target. Poolserver softwares already command miners to mine at a difficulty much lower than Bitcoin difficulty by using a higher target. This is done in order to produce "intermediate" solutions called "shares". These shares are not really intermediate solutions: they can't be continued to achieve a block solution faster. They are intermediate in the sense that the more powerful a miner is, the more shares he will create for the pool between solutions. Shares therefore provide higher granularity for accounting miners contributions. They are transmitted to the poolserver to fairly split future earnings between all involved clients, but also because any transmitted share can also be (by chance) a solution to the current Bitcoin PoW puzzle. Miners don't receive the current Bitcoin difficulty (or target) from the poolserver, they do not normally know if they found the "winning lottery ticket". The poolserver has to check for each received share if its SHA256D hash is lower than the target associated with the current Bitcoin difficulty and report it to Bitcoind. Since each secondary blockchain may have a different difficulty, the merge-mined capable poolserver has to do this check for every secondary blockchain it handles. If it finds a solution for a secondary blockchain, it sends the bitcoin block to rsdk and rskd created an SPV proof of the tag embedding and submits the proof to the secondary network. In RSK, submissions are directed to rskd which in turns adds cached transaction data and submits the new block to the RSK network.

The following table shows approximate difficulties (in terms of average nonce iterated to find a solution) of Bitcoin, RSK and shares:

Solution destination	Target interval	~Number of nonce iterations	Assumptions
Bitcoin	10 minutes	2^{68}	As of April, 2016
RSK	10 seconds	2^{62}	100% merge-mining
Mining pool share	3.3 seconds per client	2^{46}	20% hashing power, 4000 clients Ckpool software

RSK tag Embedding

The tag consist of a chunk of binary data that includes the hash of the RSK block header being mined. The tag must be uniquely located (there should not be a way to create a Bitcoin block that can be associated with two different RSK blocks). There is no accepted standard of where the tag should be stored in the block. It can be located in the coinbase field of the coinbase transaction (the first transaction of a block), in the outputs of the coinbase transaction (generally as an OP_RETURN payload), or in the remaining transactions. For RSK, the tag can only be located in the coinbase transaction. To specify a RSK block hash, the coinbase

transaction has to include the RSK tag in any part of the coinbase transaction, generally in an input or output script. The RSK tag is: *RSKBLOCK:RskBlockHeaderHash*

"RSKBLOCK:" is the ASCII string consisting of the bytes: 52 53 4b 42 4c 4f 43 4b 3a

RskBlockHeaderHash is the SHA-3 hash of the RSK Block header in binary format.

The RSK tag is meant to be included after the OP_RETURN OP_PUSHDATA1 opcodes, but this is not mandatory.

The following additional restrictions apply:

- The number of bytes immediately after RskBlockHeaderHash, up to the end of the coinbase transaction must be lower than or equal to 128 bytes.
- The trail raw bytes must not contain the binary string "RSKBLOCK:". 52 53 4b 42 4c 4f 43 4b 3a
- The probability of the RSK tag to appear by chance is negligible, but pool servers must not rule out the possibility of a rogue Bitcoin address included in the coinbase transaction having this pattern, and being used as an attack to break the validity of the merge-mined header.
- The "RSKBLOCK:" tag may appear by chance or maliciously in the ExtraNonce2 data field that is provided by miners as part of the Stratum protocol. This is not a problem as long as the poolserver adds the RSKBLOCK: tag after the ExtraNonce2 chunk.

A standard P2SH output consumes 34 bytes so the trail length limitation (the maximum number of bytes past the tag) generally means that the tag is located in the coinbase field or within the last 4 outputs of the coinbase transaction.

The tag restrictions allows RSK full node to create a compressed SPV proof that consists of:

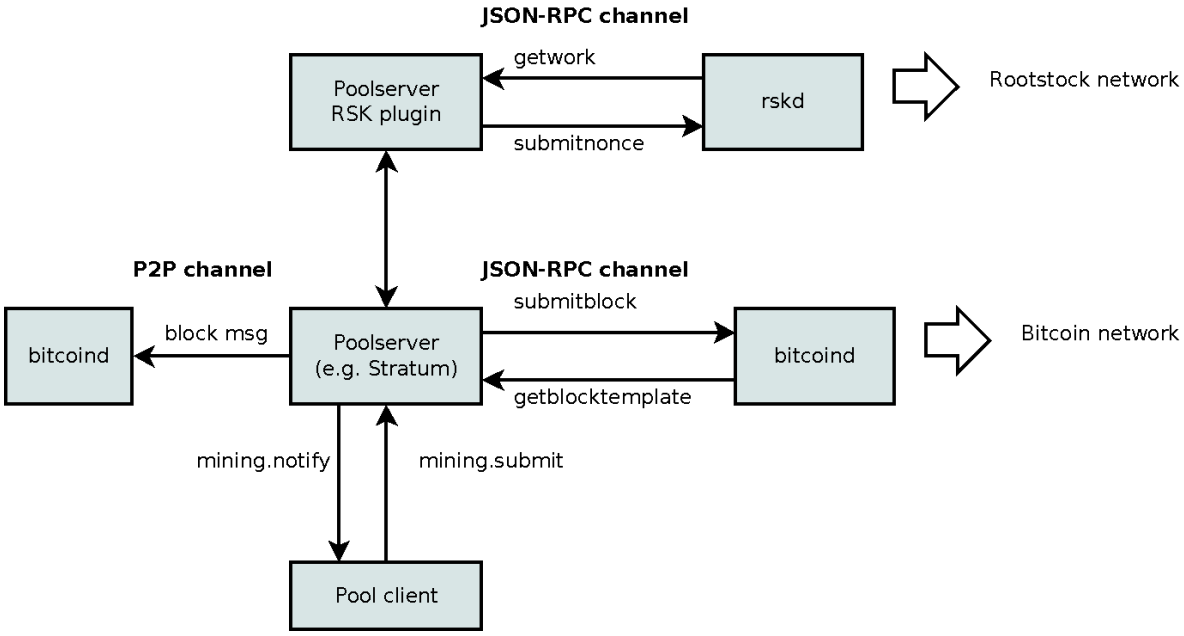
- The Bitcoin header (80 bytes)
- A Merkle Branch to the Coinbase transaction (approximately 320 bytes)
- A mid-state of SHA-256 consuming the head of the coinbase transaction (32 bytes)
- A 64 byte aligned chunk containing a trail of the coinbase transaction, including the RSK tag (max. 169 bytes). The use of a trail allows the protocol to use a proof that the trail belongs to the coinbase transaction as a free-start hash starting with the given mid-state.
- Currently the maximum size of a SPV merge-mining proof is 780 bytes.

The content of the this SPV this is opaque to the poolserver, which sends to rskd the full block. The rskd parses the Bitcoin block and extract the necessary fields to build the SPV proof.

The RskBlockHeaderHash is created by standard RSK node (rskd daemon). Rskd exposes a mining RPC-JSON interface containing a getwork command. A poolserver plugin (pool-plugin) polls rskd and maintains the latest RskBlockHeaderHash value to provide to the poolserver. If a share has enough is solved RSK block is solved, the poolserver informs the plugin, which in turns informs the rskd daemon.

RMM Plug-in Development

The following figure shows a standard poolserver architecture, including a RSK merge-mining plugin:



The PoolServer RSK plugin communicates with the RSK daemon (rskd) using two JSON-RPC methods: **mnr_getwork** and **mnr_submitBitcoinBlock**. The JSON-RPC connection port of rskd can be configured, and the default value is 4242.

The data exchange format of JSON-RPC is described here: https://github.com/ethereum/wiki/wiki/JSON-RPC.

Those methods are implemented along many others that belong to the Web3 interface, which is the standard interface to connect to rskd node. Although it's not necessary to understand web3 for developing a RSK merge-mining plugin, the full interface is defined here: https://github.com/ethereum/wiki/wiki/JavaScript-API.

Description of Methods

mnr_GetWork: This method of rskd is called by the PoolServer RSK merge-mining plugin and returns a JSON object containing the following items:

1. **blockHashForMergedMining**: Hash of the RSK block that should be merge-mined. This hash must be included after the RSKBLOCK tag. It is a 256-bit unsigned integer as an hexadecimal string
2. **Target**: Target difficulty of such block. This is a 256-bit unsigned integer as an hexadecimal string. The RSK block is solved if the hash is below or equal this target.
3. **parentBlockHash**: Parent block hash, 256-bit unsigned integer as an hexadecimal string.
4. **feesPaidToMiner**: Fees paid to the miner in the RSK block in 1/10^18 bitcoins.
5. **Notify**: This is a boolean flag that is used by rskd to alert the RSK merge-mining plugin that the new work unit should replace all previous work units as fast as possible. For example, rskd may have an internal policy that this flag will be set to 1 if the parent block hash has changed or the fees paid to the miner have increased substantially (10x) and they represent more than 20 USD. The PoolServer does not need to use this value. It can have its own policy to decide whether to notify immediately the workers or to delay the notification until more fees have been collected or a the workers need to be notified anyway because of a change in the Bitcoin parent block. The value feedPaidToMiner and parentBlockHash are informed to the merge-mining plug-in to make the best economic decision.

Sample Communication transcript:

Mining-pool call to rskd:

POST / HTTP/1.1
Authorization: Basic dXNlcjpwYXNz
Host: 127.0.0.1:4444
Content-type: application/json
Content-Length: 70

{"jsonrpc": "2.0", "method": "mnr_getWork", "params": [], "id": 4662}

Rskd response to mining-pool:
HTTP/1.1 200 OK

Content-Type: application/json-rpc
Transfer-Encoding: chunked

Server: Jetty(9.0.z-SNAPSHOT)
125

{"jsonrpc":"2.0","id":4662,"result":

{"blockHashForMergedMining":"3615d4dbc467a16d43bf35697263b730e513a5ef4dbc467a16d43bf3814e2201",

"Target":"5555555555555555555555555555555555555555555555555555555555555555",

"feesPaidToMiner":"0",

"notify":true,

"parentBlockHash":"a16d43bf3814e220124dbc467a16d43ba16d43bf3814e220124dbc467a16d43bb"}}
0

mnr_submitBitcoinBlock: This method of rskd is called by the PoolServer RSK Plugin to send a solved Bitcoin block (at the RSK target difficulty) to be processed by rskd. However, not the full block is sent, only an SPV proof. The contents of ther SPV proof is defined in the "RSK Merge Mining (RMM) and Bitcoin Interoperability" document. Rskd will extract the merge-mining RSK block header hash from the block coinbase and submit it to the RSK network. In the future the RSK Plugin will send only an SPV proof, instead of the full block.

Sample Communication transcript:

Mining-pool call to rskd:

POST / HTTP/1.1
Authorization: Basic dXNlcjpwYXNz
Host: 127.0.0.1:4444
Content-type: application/json
Content-Length: 612

{"jsonrpc": "2.0", "method": "**mnr_submitBitcoinBlock**", "params": ["0100003019f3857b30a344e4ca0e83b836143bf3561215e0428dcc6a540d5fe701000000808f90c2ef8968087703158618a4163759258ace00c2c5602469c1ae441eaec873724857ffff7f2001fd76d00101000000010000000000000000000000000000000000000000000000000000000000000000ffffffff300258010004737248570426 9d78060ccb6848570000000000000000a636b706f6f6c0d2f6d696e656420627920636b2fffffffff02807c814a000000001976a914d754fa97263b730e513a5ef8

14e2204e1c6a4aef88ac00000000000000002b6a524f4f5453544f434b3a6b196f3481b6f5e71373
3eec0270e6b8205d4dbc467a16d1504777f972658fa100000000"], "id": 5093}

Rskd response to mining-pool:

HTTP/1.1 200 OK
Content-Type: application/json-rpc
Transfer-Encoding: chunked
Server: Jetty(9.0.z-SNAPSHOT)

29
{"jsonrpc":"2.0","id":5093,"result":null}
0

Basic HTTP authentication

Production versions of Rskd will allow to secure the connection using basic HTTP authentication (https://en.wikipedia.org/wiki/Basic_access_authentication). Therefore the merge mining plugin should allow the configuration of user and a password strings. For greater security, and in case where rskd and the PoolServer are not located in the same machine, the connection should be encapsulated in a ssh tunnel.

Past Developments

The RSK team has developed a fully working RMM Plugin for the CoiniumServ PoolServer, ckpool and eloipool softwares. These reference implementations are provided by RSK on request. Ckpool implementation is is optimized to provide negligible performance penalty on Bitcoin mining. On the contrary, CoiniumServ PoolServer plug-in implementation is provided for clarity, but its performance has not been fully tested.

Developing a Merge-mining Plugin

Generally a plug-in consist of 150-300 lines of code, without considering RSK reward accounting and splitting modules.

The following configuration options are recommended:

- **PollPeriod**: This is the period of time [in seconds] which the plugin should poll rsks using getwork for updated units of work. If PollPeriod is set to zero, then the RSK plugin should only poll for a new unit of work when creating new units of work for Bitcoin (e.g. every 3.3 seconds).

- **NotifyPolicy**: if this is 1, the poolserver should broadcast new work units when the notify flag in the getwork return object is true. If this is 2, it should use a simple logic to decide: if the parent block has changed, then notify. If this is 0, no immediate notification will be performed and the next RSK work unit will be send along the next Bitcoin work unit.

Suggested Efficiency Requirements

The code should be optimized such that:

- Blocking the communications with the rskd application does not block any main functionality of the PoolServer (such as creating new Bitcoin work units and broadcasting them).
- Because of RSK uses the DECOR+ protocol, the PoolServer should keep processing ALL solutions found by the pool clients, even if the RSK parent hash changed.
- If a block solution is simultaneously a Bitcoin solution and a RSK solution, the application should make sure the notification to the rskd does not delay the notification to bitcoind. One possibility is to perform the rskd notification AFTER the bitcoind notification.
- The PoolServer should not delay the reception of new work units from bitcoind by the reception of work units from rskd. In other words, those tasks should not be performed by a single blocking thread.
- The merge-mining code should not consume excessive CPU or memory resources (such as using empty loops for sleeping, or leaking memory).
- If a new Bitcoin block is found by an external miner, and the Poolserver is notified, any pending notifications to miners should be aborted and a new notification round should begin immediately. This should not be the case if a new RSK block is found by an external miner, but miners are currently being notified of a new parent block hash. In other words, the Poolserver should always prioritize notifications to miners related to Bitcoin parent block changes over any other notification message. This optimization is not mandatory and depends on the poolserver architecture. Most poolserver softwares work asynchronously and therefore can notify all client miners in a short period of time, so removing previously enqueued messages or aborting notification loops does not yield a meaningful gain.

The aim is always that the operation as a Bitcoin mining pool server is not affected by the overhead of handling the RSK merge-mining duties.

Testing

We recommend having a test-suite to learn about and tune Poolserver efficiency. Poolserver should be instrumented with logging info containing timestamps with millisecond precision. Information should be logged in a format which can be easily interpreted by a program to compute average times. The standard events that should be logged are:

- New block parent detected in bitcoind (which means a new Bitcoin work unit must be built). This is because bitcoind is queried using getblocktemplate and not getwork.
- New work unit received from bitcoind
- New work unit sent to a peer
- Work unit sent to the last peer working on an old work unit.
- Share received by a client (can be a Bitcoin solution, RSK solution, or none of them)
- Bitcoin solution received by a client
- RSK solution received by a client
- RSK SPV will be sent to rskd
- RSK SPV proof sent (after the event)
- Bitcoin block will be sent to bitcoind
- Bitcoin block sent to bitcoind (after the event)

The test-suit should stress-test the Pool server software during merge-mining by accepting connections from 1000 simultaneous clients (either 1000 cpuminers or 999 null-miners and 1 cpuminer). The test should be run for at least 6 hours. For the test , the following conditions should be met:

- The average time between a getblocktemplate is called to bitcoind (a new work unit is created) and the time the last client is sent the new work unit should not be higher than 15 msec.
- The average time between a Bitcoin solution is received by a client and the Bitcoin block is sent to bitcoind should not be higher than 15 msec.

-- end of document --