Harmony RPC 加速阶段性讨论

时间：2022/04/13

> **主题：**
>
> - Harmony 节点 RPC 测试数据
> - 测试数据分析
> - 给出的解决方案
> - 基于负载均衡的优化
> - 方案转折点
> - 目前的进展

Harmony 节点 RPC 测试数据

Harmony 节点的RPC测试数据测了两版，分别有产出文档和测试数据。

截取部分测试数据 总请求50， 并发15：

hmyv2_blockNumber

```
https://a.api.s0.t.hmny.io
hmyv2_blockNumber

Complete requests:      50
Failed requests:        0
Write errors:           0
Time per request:       1299.648 [ms] (mean)
```

hmyv2_getBlockSigners

```
https://a.api.s0.t.hmny.io
hmyv2_getBlockSigners

Complete requests:      50
Failed requests:        0
Write errors:           0
Time per request:       1217.336 [ms] (mean)
```

hmyv2_getBlocks

```
https://a.api.s0.t.hmny.io
hmyv2_getBlocks

Complete requests:      50
Failed requests:        0
Write errors:           0
Time per request:       2315.090 [ms] (mean)
```

hmyv2_getBalanceByBlockNumber

```
https://a.api.s0.t.hmny.io
hmyv2_getBalanceByBlockNumber

Complete requests:      50
Failed requests:        0
Write errors:           0
Time per request:       1394.852 [ms] (mean)
```

hmyv2_getValidatorInformation

```
https://a.api.s0.t.hmny.io
hmyv2_getValidatorInformation

Complete requests:      50
Failed requests:        0
Write errors:           0
Time per request:       1197.244 [ms] (mean)
```

hmyv2_call

```
https://a.api.s0.t.hmny.io
hmyv2_call

Complete requests:      50
Failed requests:        0
Write errors:           0
Time per request:       1187.868 [ms] (mean)
```

net_peerCount

```
https://a.api.s0.t.hmny.io
net_peerCount

Complete requests:      50
Failed requests:        0
Write errors:           0
Time per request:       1171.618 [ms] (mean)
```

hmyv2_getNodeMetadata

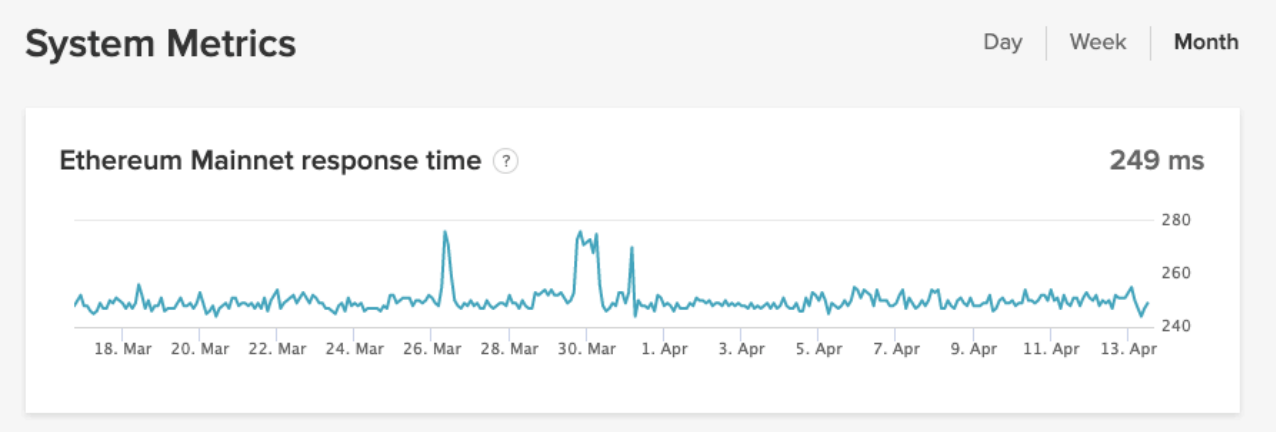
```
https://a.api.s0.t.hmny.io
hmyv2_getNodeMetadata

Complete requests:      50
Failed requests:        0
Write errors:           0
Time per request:       1437.167 [ms] (mean)
```

```
https://rpc.s0.t.hmny.io
hmyv2_getCurrentTransactionErrorSink

Complete requests:      50
Failed requests:        48
   (Connect: 0, Receive: 0, Length: 48, Exceptions: 0)
Write errors:           0
Non-2xx responses:      50
================================================================

https://rpc.s0.t.hmny.io
hmyv2_getCurrentUtilityMetrics

Complete requests:      50
Failed requests:        1
   (Connect: 0, Receive: 0, Length: 1, Exceptions: 0)
Write errors:           0
Non-2xx responses:      50
```

测试数据分析

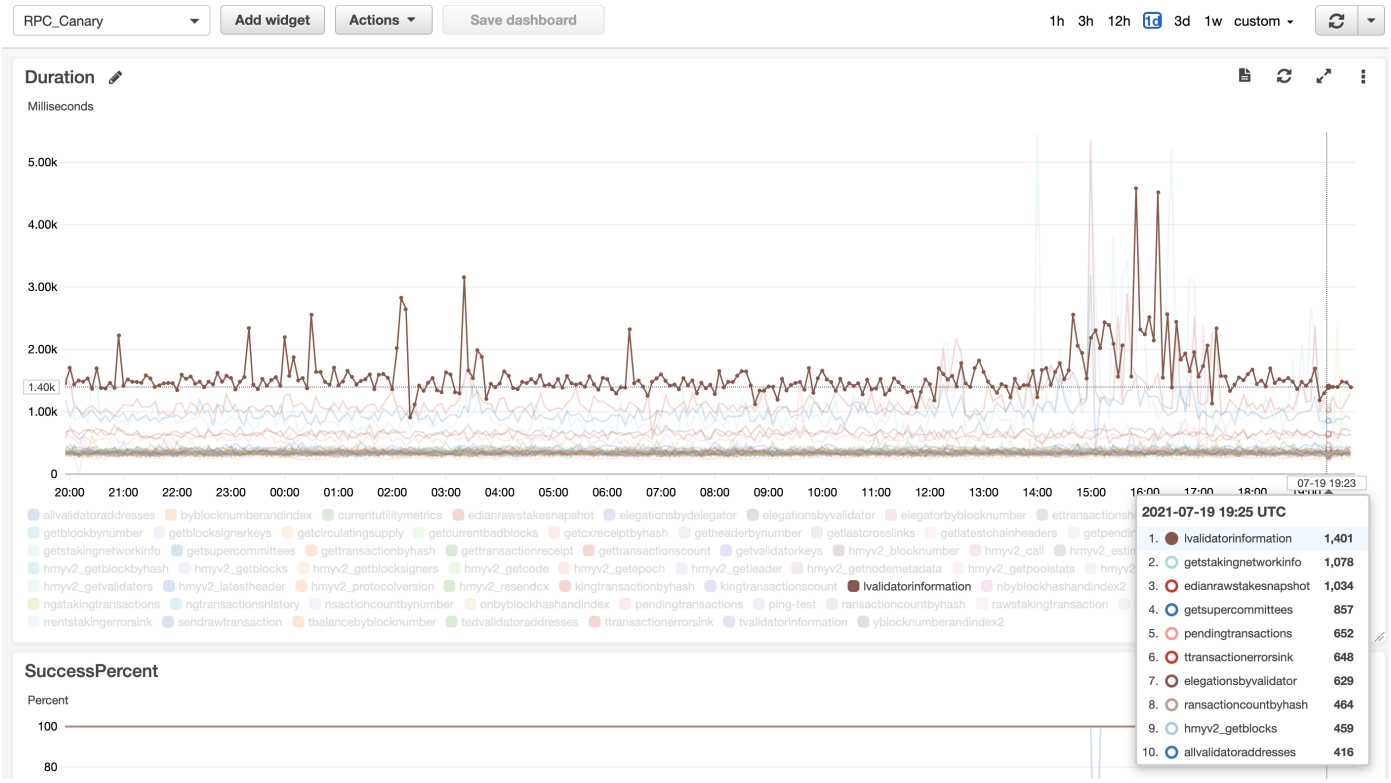
由测试数据可以看出，从节点提供RPC服务的 `稳定性` `请求速率（RPS）` 两个维度来分析。

参考第三方团队 Infura 提供 Ethereum 的 RPC 服务，Infura 在节点服务的稳定性和请求延迟如下图：



从图中可以看出，Infura提供的服务的稳定性和请求延迟上都有比较好的表现。

我们从 Harmony 官方的 Github 上也可以看出，有ISSUE也指出RPC请求过慢的问题（Closed ISSUE Link ）：



不过此ISSUE Closed，但从最近的测试结果看，还有优化的空间。

给出的解决方案

1. 堆硬件，在AWS这种云服务商上部署节点，可以考虑加大内存、使用Enhanced SSD来解决；
2. 在提供服务的内网，使用nginx做负载均衡来保证服务的可用性和较低的请求延时。

我们尝试过负载均衡的策略，在内网几台性能机上部署了S0分片的节点，以Validator的模式运行（非Explorer模式）。结构图如下：

在此基础上，我们测试了服务的可用性和性能。测试数据如下：

hmyv2_blockNumber

```
https://hmyapis0.metamemo.one
hmyv2_blockNumber

Complete requests:      50
Failed requests:        0
Time per request:       148.327 [ms] (mean)
Time per request:       9.889 [ms] (mean, across all concurrent requests)
```

hmyv2_getBlockSigners

```
https://hmyapis0.metamemo.one
hmyv2_getBlockSigners

Complete requests:      50
Failed requests:        0
Time per request:       109.220 [ms] (mean)
Time per request:       7.281 [ms] (mean, across all concurrent requests)
```

hmyv2_getBlocks

```
https://hmyapis0.metamemo.one
hmyv2_getBlocks

Complete requests:      50
Failed requests:        0
Time per request:       1562.746 [ms] (mean)
Time per request:       104.183 [ms] (mean, across all concurrent requests)
```

hmyv2_getBalanceByBlockNumber

```
https://hmyapis0.metamemo.one
hmyv2_getBalanceByBlockNumber

Complete requests:      50
Failed requests:        0
Time per request:       140.976 [ms] (mean)
Time per request:       9.398 [ms] (mean, across all concurrent requests)
```

hmyv2_getValidatorInformation

```
https://hmyapis0.metamemo.one
hmyv2_getValidatorInformation

Complete requests:      50
Failed requests:        0
Time per request:       159.583 [ms] (mean)
Time per request:       10.639 [ms] (mean, across all concurrent requests)
```

hmyv2_call

```
https://hmyapis0.metamemo.one
hmyv2_call

Complete requests:      50
Failed requests:        0
Time per request:       134.425 [ms] (mean)
Time per request:       8.962 [ms] (mean, across all concurrent requests)
```

net_peerCount

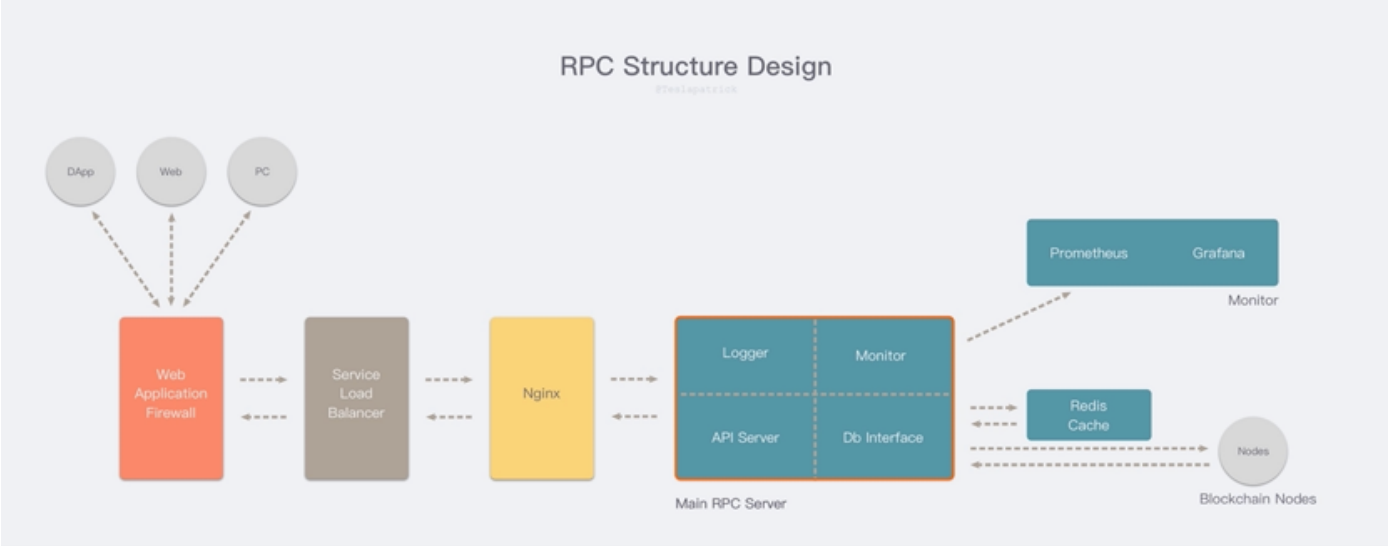
```
https://hmyapis0.metamemo.one
net_peerCount

Complete requests:      50
Failed requests:        0
Time per request:       105.284 [ms] (mean)
Time per request:       7.019 [ms] (mean, across all concurrent requests)
```

hmyv2_getNodeMetadata

```
https://hmyapis0.metamemo.one
hmyv2_getNodeMetadata

Complete requests:      50
Failed requests:        34
   (Connect: 0, Receive: 0, Length: 34, Exceptions: 0)
Time per request:       186.314 [ms] (mean)
```

基于负载均衡的优化

基于讨论和问题分析的结果，针对Harmony的RPC 加速结构如下图所示。



DApp、Web、PC端的请求，经过Load Balancer，最终会到达我们的RPC Server。

其中的Db Interface主要是请求数据。数据的来源可以是Redis Cache、Nodes或是SQL Server等。由于目前的瓶颈主要在读取链上的数据，我们可以使用Redis Cache 存储一些量大且不变的数据，如果是Balance等经常更新的数据，不适合存储在Redis中，同时要考虑Redis数据的读写分离，增加Cache系统的稳定性。

Redis Cache的缓存策略，最贴近目前需求的是Cache Aside Pattern：

- Cache 未命中：DApp先从cache取链上数据，如果没有得到，则从Nodes中取数据，成功返回后放到Redis缓存中。
- Cache 命中：DApp从cache中取链上数据，取到后返回。

基于以上的结构，我们主要实现了：

1. RPC Server （RPC Proxy）的 节点Endpoint筛选 负载均衡策略开发；
    - 基于权重的负载均衡策略

```
type WeightMap struct {
  len   int
  lock  sync.RWMutex
  Nodes []*nodeWithWeight
}

type nodeWithWeight struct {
  rpc             rpc.RPCClient
  initWeight      int64
  currentWeight   int64
  effectiveWeight int64
}
```

- 一致性Hash

```
type ConsistentHashMap struct {
  hash    func(data []byte) uint32
  hashMap map[uint32]rpc.RPCClient
  keys    uint32Slice
  replica int
  len     int
  lock    sync.RWMutex
}

type uint32Slice []uint32
```

2. 中转RPC的请求到节点Endpoint

```
func (l *ApiLogic) handleApiMessage(id uint64, method string, params jsonrpc.Params,
loadBalance balancing.LoadBalance) (resp *types.Response, err error) {
  // todo: 降低圈复杂度!
  switch method {
  case "getBalance":
    var address string
    params.UnmarshalSingleParam(0, &address)
    if l.useCache() {
      balance, err := l.svcCtx.Cache.GetBalance(address)
      if balance != nil {
        raw, _ := balance.MarshalJSON()
```

```
            return l.SuccessResponse(id, raw), err
          }
        }
        // get balance from endpoint
        res, err := loadBalance.Get("getBalance").
    (*hmy.HmyRpcClient).GetBalance(context.Background(), address)
        if res.Error != nil {
          return l.ErrorResponse(id, res.Error), nil
        }

        var r *big.Int
        if res.Result != nil {
          err = json.Unmarshal(*res.Result, &r)
        }
        // write to cache
        l.svcCtx.Cache.SetBalance(address, r)

        return l.SuccessResponse(id, *res.Result), nil
      }

        case "getBalanceByBlockNumber":

        ...
        ...
        ...
    }
```

3. Redis 缓存链上数据的功能开发

```
type Config struct {
  Endpoint string `json:"endpoint"`
  Password string `json:"password"`
  Database int    `json:"database"`
  PoolSize int    `json:"poolSize"`
}

type RedisClient struct {
  client  *redis.Client
  prefix  string
  timeout time.Duration
}

func (r *RedisClient) SetBalance(address string, balance *big.Int) error {
```

```
    pipe := r.client.TxPipeline()
    defer pipe.Close()
    // set balance
    pipe.HMSet(context.Background(), r.formatKey("balanceLatest", address), map[string]interface{}
{
    "latest": balance.String(),
    })
    // set expire time
    pipe.Expire(context.Background(), r.formatKey("balanceLatest", address), r.timeout)
    _, err := pipe.Exec(context.Background())
    return err
}
```

方案转折点

在测试代码时用官方的 `net/http` 包替换为 `fasthttp`，在第一次连接时建立连接比较久（不同的endpoint不同时间），fasthttp在 `DefaultMaxIdleConnDuration = 10 * time.Second` 内不会释放TCP连接，这样在 `DefaultMaxIdleConnDuration` 时间内，可以再次发起请求，而不需要重新建立连接。

这样的连接的请求情况如图：

图。

同时fasthttp默认的连接数为 `DefaultMaxConnsPerHost = 512`，这样在并发测试下，会有更好的表现。

基于fasthttp测试的情况，可以把请求后端endpoint的连接统一用 `连接池` 替代 `net/http`。因为经过nginx负载均衡后，RPC Server（RPC Proxy）跟后端Endpoint之间不需要频繁建立 / 断开 TCP连接。

这样从RPC请求过来，经过nginx、RPC Server的负载均衡，再到Endpoint的可以如下图：

图。

目前的进展

目前在RPC加速主要的开发如下：

1. Endpoint端的负载均衡；
2. 连接池；
3. Redis Cache 缓存无状态的链上数据。

基于开发1、2功能的基础上的测试数据：

hmyv2_blockNumber

```
http://103.39.231.220:8888/api/v1/hmy/v2
hmyv2_blockNumber

Complete requests:      50
Failed requests:        0
Time per request:       244.884 [ms] (mean)
Time per request:       16.326 [ms] (mean, across all concurrent requests)
```

hmyv2_getBlockSigners

```
http://103.39.231.220:8888/api/v1/hmy/v2
hmyv2_getBlockSigners

Complete requests:      50
Failed requests:        0
Time per request:       265.416 [ms] (mean)
Time per request:       17.694 [ms] (mean, across all concurrent requests)
```

hmyv2_getBlocks

```
http://103.39.231.220:8888/api/v1/hmy/v2
hmyv2_getBlocks

Complete requests:      50
Failed requests:        0
Time per request:       1611.972 [ms] (mean)
Time per request:       107.465 [ms] (mean, across all concurrent requests)
```

hmyv2_getBalanceByBlockNumber

```
http://103.39.231.220:8888/api/v1/hmy/v2
hmyv2_getBalanceByBlockNumber

Complete requests:      50
Failed requests:        0
Time per request:       105.700 [ms] (mean)
Time per request:       7.047 [ms] (mean, across all concurrent requests)
```

hmyv2_getValidatorInformation

```
http://103.39.231.220:8888/api/v1/hmy/v2
hmyv2_getValidatorInformation

Complete requests:      50
Failed requests:        0
Time per request:       108.739 [ms] (mean)
Time per request:       7.249 [ms] (mean, across all concurrent requests)
```

hmyv2_call

```
http://103.39.231.220:8888/api/v1/hmy/v2
hmyv2_call

Complete requests:      50
Failed requests:        0
Time per request:       119.941 [ms] (mean)
Time per request:       7.996 [ms] (mean, across all concurrent requests)
```

net_peerCount

```
http://103.39.231.220:8888/api/v1/hmy/v2
net_peerCount

Complete requests:      50
Failed requests:        0
Time per request:       88.574 [ms] (mean)
Time per request:       5.905 [ms] (mean, across all concurrent requests)
```

hmyv2_getNodeMetadata

```
http://103.39.231.220:8888/api/v1/hmy/v2
hmyv2_getNodeMetadata

Complete requests:      50
Failed requests:        0
Time per request:       85.439 [ms] (mean)
Time per request:       5.696 [ms] (mean, across all concurrent requests)
```

基于Redis Cache的测试数据：

待补充。