

By 区块链 keep 哥

结论:

- 目前 EOS 的 Ram 大小是写死在程序里的, 所以不升级软件是绝对不可能扩容的, 市场上各种扩容的风传完全站不住脚
- Bancor relay weight 的从 0.5 变化 50 不会改变 ram 的价格趋势, 计算过程我上传到 <https://pan.baidu.com/s/1U3LCAVDUT5IHHDGPGDp0bw>。有兴趣的同学可以下载检验一下, 如果你觉得有计算过程有疑问可以和我联系。
- Ram 占用率 vs 价格趋势如下, 占用率达到 87.78%时, ram 的价格是 1eos/Kbyte; 当占用达到 90.01%时, ram 的价格是 1.5 eos/Kbyte.

Bancor weight = 0.5时, ram 占用率 vs 价格趋势		Bancor weight = 50时, ram 占用率 vs 价格趋势	
行标签	求和项:平均价格(Eos/KB)	行标签	求和项:平均价格(Eos/KB)
80.00%	0.373641016	80.00%	0.373641016
85.01%	0.665573182	85.01%	0.665573182
87.78%	1.001819136	87.78%	1.001819136
90.00%	1.497536378	90.00%	1.497536378
90.01%	1.500518026	90.01%	1.500518026

详细分析:

- 目前 EOS 的 Ram 大小是写死在程序里的, 所以不升级软件是绝对不可能扩容的, 市场上各种扩容的风传完全站不住脚

```
struct eosio_global_state : eosio::blockchain_parameters {
    uint64_t free_ram()const { return max_ram_size - total_ram_bytes_reserved;
    uint64_t max_ram_size = 6411*1024 * 1024 * 1024; |
    uint64_t total_ram_bytes_reserved = 0;
    int64_t total_ram_stake = 0;

    block_timestamp_t last_producer_schedule_update;
    uint64_t last_pervote_bucket_fill = 0;
    int64_t pervote_bucket = 0;
    int64_t perblock_bucket = 0;
    uint32_t total_unpaid_blocks = 0; // all blocks which have been produced but not paid
    int64_t total_activated_stake = 0;
    uint64_t thresh_activated_stake_time = 0;
    uint16_t last_producer_schedule_size = 0;
    double total_producer_vote_weight = 0; // the sum of all producer votes
    block_timestamp_t last_name_close;

    // explicit serialization macro is not necessary, used here only to improve compilation time
    EOSLIB_SERIALIZE_DERIVED( eosio_global_state, eosio::blockchain_parameters,
        (max_ram_size)(total_ram_bytes_reserved)(total_ram_stake)
        (last_producer_schedule_update)(last_pervote_bucket_fill)
        (pervote_bucket)(perblock_bucket)(total_unpaid_blocks)(total_activated_stake)(thresh_activated_stake_time)
        (last_producer_schedule_size)(total_producer_vote_weight)(last_name_close) )
};
```

Ram 的总量是写死在程序里的, 所以不升级软件是绝对不可能扩容的, 所以市场上各种扩容的风传完全站不住脚。

如果要按照 BM 说的, 每产生一个区块, 扩容一点点, 这个代码的逻辑会很复杂, 正常情况下, 短期(一周)内没有上线的可能性。

如果各位写过程序, 对于这种写法, 是不是想开骂? 😊。

- Eos 使用 rammarket 来记录可用内存量和货币量的关系

在 eosio.system 合约下有一张表 rammarket, 它用来记录内存量和货币量的比值。

使用 cleos get table eosio eosio rammarket, 可以看到其中的数据。

在 2018/7/6 10:43 分查询, 可以看到: base.balance 记录了当前未被购买的 ram 大约有 12G; 用于购买 ram 的 eos 大约有 530 万。

全网预览 请id: aal78 (版本: v0.7.0.247)

区块数量 4,393,596 交易数量 2,223,041 消息数量 3,409,393 帐户数量 255,145

30.40% (101,957,343.36 EOS)

投票比例 100%

价格走势

当前价格 8.86 USD -0.01%

当前市值 7,940,771,689.00 USD

5.10 9.00 8.90 8.80 8.70

22:00 07-06 08:00 10:00 14:00 18:00

RAM兑换 关于RAM

当前EOS/美元汇率

5,300,440.9831 EOS

12,661,142.961 KiB

帐户竞拍 查看更多竞拍

con 75000.0000 EOS

bank 2056.0000 EOS

one 489.0000 EOS

```

root@iZm5e6ig9oqpk7bxfmtl1kZ:~# cleos get table eosio eosio rammarket
{
  "rows": [
    {
      "supply": "10000000000.0000 RAMCORE",
      "base": {
        "balance": "12953920358 RAM",
        "weight": "0.500000000000000000"
      },
      "quote": {
        "balance": "5304978.7633 EOS",
        "weight": "0.500000000000000000"
      }
    },
    {
      "supply": "10000000000.0000 RAMCORE",
      "base": {
        "balance": "12953920358 RAM",
        "weight": "0.500000000000000000"
      },
      "quote": {
        "balance": "5304978.7633 EOS",
        "weight": "0.500000000000000000"
      }
    }
  ],
  "more": false
}

```

红色: 未被分配的内存量

黄色: 用于购买 ram 的 eos 数量

```

root@iZm5e6ig9oqpk7bxfmtl1kZ:~# cleos -u http://mainnet.genereos.io get table eosio eosio rammarket
{
  "rows": [
    {
      "supply": "10000000000.0000 RAMCORE",
      "base": {
        "balance": "12979869510 RAM",
        "weight": "0.500000000000000000"
      },
      "quote": {
        "balance": "5294373.1753 EOS",
        "weight": "0.500000000000000000"
      }
    }
  ],
  "more": false
}

```

rammarket 是在首次调用 eos 系统自带的智能合约，初始化时会将可用内存(64G)，以及购买内存的 eos 记录下来,需要说明一点，按照代码逻辑，初始化的时候有 100 万的 eos 用于购买 OKB 的内存。

```

system_contract::system_contract( account_name s )
: native(s),
_voters(_self, _self),
_producers(_self, _self),
_global(_self, _self),
_rammarket(_self, _self)
{
  //print( "construct system\n" );
  _gstate = _global.exists() ? _global.get() : get_default_parameters();

  auto itr = _rammarket.find(S(4, RAMCORE));

  if( itr == _rammarket.end() ) {
    auto system_token_supply = eosio::token(N(eosio.token)).get_supply(eosio::symbol_type(system_token_symbol).name()).amount;
    if( system_token_supply > 0 ) {
      itr = _rammarket.emplace( _self, [&]( auto& m ) {
        m.supply.amount = 1000000000000001; //初始化supply, 这是Ram 和EOS 换算的一个中间量, 100亿, 小数点后有4位.
        m.supply.symbol = S(4, RAMCORE);
        m.base.balance.amount = int64_t(_gstate.free_ram()); //base.amount 记录了目前还未分配的内存量, EOS启动的那一刻是64G.
        m.base.balance.symbol = S(0, RAM);
        m.quote.balance.amount = system_token_supply / 1000; // 100万 eos
        m.quote.balance.symbol = CORE_SYMBOL;
      });
    }
  }
  else {
    //print( "ram market already created" );
  }
}
} // end system_contract

```

调用 buyram 函数来计算 quant 个 eos, 能卖多少 byte 的 ram.

```
//payer:付款人
//receiver:接收人
//quant: eos的数量
void system_contract::buyram( account_name payer, account_name receiver, asset quant )
{
    require_auth( payer );
    eosio_assert( quant.amount > 0, "must purchase a positive amount" );

    auto fee = quant;
    fee.amount = ( fee.amount + 199 ) / 200; /// .5% fee (round up)
    // fee.amount cannot be 0 since that is only possible if quant.amount is 0 which is not allowed by the assert above.
    // If quant.amount == 1, then fee.amount == 1,
    // otherwise if quant.amount > 1, then 0 < fee.amount < quant.amount.
    auto quant_after_fee = quant;
    quant_after_fee.amount -= fee.amount;
    // quant_after_fee.amount should be > 0 if quant.amount > 1.
    // If quant.amount == 1, then quant_after_fee.amount == 0 and the next inline transfer will fail causing the buyram action to fail.

    INLINE_ACTION_SENDER(eosio::token, transfer)( N(eosio.token), {payer,N(active)},
    { payer, N(eosio.ram), quant_after_fee, std::string("buy ram") } );

    if( fee.amount > 0 ) {
        INLINE_ACTION_SENDER(eosio::token, transfer)( N(eosio.token), {payer,N(active)},
        { payer, N(eosio.ramfee), fee, std::string("ram fee") } );
    }
}

int64_t bytes_out;
const auto& market = _rammarket.get(S(4,RAMCORE), "ram market does not exist");
_rammarket.modify( market, 0, [&]( auto& es ) {
    bytes_out = es.convert( quant_after_fee, S(0,RAM) ).amount;
});

eosio_assert( bytes_out > 0, "must reserve a positive amount" );

_gstate.total_ram_bytes_reserved += uint64_t(bytes_out);
_gstate.total_ram_stake += quant_after_fee.amount;

user_resources_table userres( _self, receiver );
auto res_itr = userres.find( receiver );
if( res_itr == userres.end() ) {
    res_itr = userres.emplace( receiver, [&]( auto& res ) {
        res.owner = receiver;
        res.ram_bytes = bytes_out;
    } );
} else {
    userres.modify( res_itr, receiver, [&]( auto& res ) {
        res.ram_bytes += bytes_out;
    } );
}

set_resource_limits( res_itr->owner, res_itr->ram_bytes, res_itr->net_weight.amount, res_itr->cpu_weight.amount );
}
```

扣除 0.5% 的手续费

调用 covert 函数计算扣完手续费之后的 eos 能卖多少 ram

更新系统资源表

3. Covert 函数实现了 bancor 算法, 分析的结果是 bancor relay weight 的从 0.5 到 50 不会改变价格趋势。



