可以将多个行中的多个列用一个merge请求传过去

```
/***This is new.**********/
message MergeRequestNew {
  message Transaction{
    message Row{
      string tableName = 1;
      bytes key = 2;

      //原来直接copy一行数据
      //bytes data= 3;

      message Column{
        uint64 id = 1;
        bytes value = 2;
      }

      repeated Column col = 3;
    }
    uint64 TxnID = 1;
    repeated Row row = 2;
   uint64 startts = 3;
   uint64 committs = 4;
  }
  repeated Transaction Txn = 1;
  string request_address = 2;
}
```

相当于

```
class mergerequest{
    vector<Transaction> trans;
    string request_adress;
}
class Transaction{
    vector<Row> rows;
    uint64 TxnId;
    uint64 startts;
    uint64 commits;
}
class Row{
```

```
    string tablename;
    bytes key;
    vector<Column> cols;
}
class Column{
    uint64 id;
    bytes values;
}
```

存储结构，但是protobuf序列化的空间会比正常数据更小，在RequestMergeThread中将数据序列化后，再使用zmq传序列化后的数据给其他节点，在ProcessMergeThread中，将数据解析，再进行相应的操作

发送方通过查看local_changeset有哪些事务进行了修改数据库的操作并将数据按操作保存到MergeRequest类的实例中，如果是MOT::INS操作那么将把一行内所有的列数据保存记录，如果是MOT::WR操作，则使用bitmap判断一行内的每一列数据有没有被更改，更改了则记录列号和数据，对应的是column中的id和value，将数据进行保存到MergeRequest中，当保存好local_changeset中所有修改的信息后，对数据进行serialize，然后通过zmq框架进行通信，绑定目的ip然后发送数据。

接收方通过一个线程监听绑定的ip进行判断是否有远程信息，当收到消息时先要对消息进行反序列化因为在传输消息时是经过序列化的，直接使用protobuf中的反序列化的函数即可，处理后的数据是与我们设计的protobuf的结构是相同的，我们可以将所有信息取出，进行验证每个事务是否可以提交，因为有对应的表名、行的key和列的id所以可以直接将对应的数据修改，最后将可以提交的数据写到数据库即可。