HBase/Phoenix RPC Scheduler Framework

This document describes HBase RPC framework and how Phoenix extends it. It also discusses the challenges and gaps in the framework and the potential solutions for PHOENIX-7370.

Background:

Every read/write request requires an up-to-date schema of the table before making RPC requests. Every request makes an RPC call to the regionserver serving SYSTEM.CATALOG to fetch the PTable object, with the only exception being table schema param: `UPDATE_CACHE_FREQUENCY`. If UPDATE_CACHE_FREQUENCY is not ALWAYS, the client can cache the PTable object for the duration specified by UPDATE_CACHE_FREQUENCY. In this case, not every query ends up making an RPC call to the regionserver serving SYSTEM.CATALOG region(s).

We have seen several latency issues for tenant queries running on a large multi-tenant table in production. These problems are associated with the fact that Coproc MetaDataEndpointImpl that is primarily available on SYSTEM.CATALOG region serving regionservers, takes write-lock on <tenant-id, schema-name, table-name> tuple before scanning SYSTEM.CATALOG or retrieving PTable object from server side cache. Upserts or Selects on tenant based views require PTable object of the parent base table and hence, all multi-tenant read-write requests require base table PTable object. This can significantly cause performance issues as we hold getTable() calls with write-lock for shared base tables by all multi-tenant requests. As part of PHOENIX-6066, the write-lock has been replaced with the read-lock because the write-lock taken by getTable() calls do not perform any schema changes for the given table.

HBase RPC framework:

HBase uses RPC (Remote Procedure Call) framework for all the wire communication among its components e.g. client to server (client to master daemon or client to regionservers) as well as server to server (master to regionserver, regionserver to regionserver) communication. HBase RPC uses Google's Protocol Buffers (protobuf) for defining the structure of messages sent between clients and servers. Protocol Buffers allow efficient serialization and deserialization of data, which is crucial for performance. HBase defines service interfaces using Protocol Buffers, which outline the operations that clients can request from HBase servers. These interfaces define methods like get, put, scan, etc., that clients use to interact with the database.

HBase also provides Coprocessors. HBase Coprocessors are used to extend the regionservers functionalities. They allow custom code to execute within the context of the regionserver during specific phases of the given workflow, such as during data reads

(preScan, postScan etc), writes (preBatchMutate, postBatchMutate etc), region splits or even at the start or end of regionserver operations. In addition to being SQL query engine, Phoenix is also a Coprocessor component. RPC framework using Protobuf is used to define how coprocessor endpoints communicate between clients and the coprocessors running on the regionservers.

RPC priority and dedicated thread-pools:

HBase RPC framework allows clients to set priority for the given RPC call. This priority can be set in general for any RPC calls or RPC calls for specific tables:

```java
 /**
  * Set the priority for this operation.
  * @param priority Priority for this request; should fall
roughly in the range
  *                 {@link HConstants#NORMAL_QOS} to {@link
HConstants#HIGH_QOS}
  */
 void setPriority(int priority);

 /**
  * Set the priority for this operation.
  * @param tn Set priority based off the table we are going
against.
  */
 void setPriority(final TableName tn);
```

HBase uses several RPC Scheduler implementations for clients to choose the right priority for RPC call execution:

- FifoRpcScheduler
- MasterFifoRpcScheduler
- SimpleRpcScheduler

By default, regionservers use SimpleRpcScheduler, which is provided by SimpleRpcSchedulerFactory.

For each RpcScheduler, corresponding RpcSchedulerFactory is used to provide the RpcScheduler implementation:

- FifoRpcSchedulerFactory
- MasterFifoRpcSchedulerFactory
- SimpleRpcSchedulerFactory

HBase SimpleRpcScheduler provides separate thread-pools for specific workflows:

- **RpcServer.priority.FPBQ.Fifo.handler** for high priority RPC call execution
- **RpcServer.metaPriority.FPBQ.Fifo.handler** for hbase:meta region transition related RPC calls
- **RpcServer.bulkLoad.FPBQ.Fifo.handler** for bulk load workloads
- **RpcServer.replication.FPBQ.Fifo.handler** for replication RPC calls
- **RpcServer.default.FPBQ.Fifo.handler** for normal FIFO RPC queue based execution

By default, priority of any RPC call is set to 0 and it goes to default.FPBQ handler pool. The highest priority that HBase recognizes is 200. High priority requests such as meta transition or any system table read/write are served with priority.FPBQ or metaPriority.FPBQ handler pools.

Phoenix leverages this concept to provide its own RpcSchedulerFactory implementation and its own priority as well as handler thread-pools.

- **RpcServer.Metadata.Fifo.handler** for Client to Server RPC calls for Phoenix system tables
- **RpcServer.Index.Fifo.handler** for RPC calls to Index tables
- **RpcServer.ServerSide.Fifo.handler** for Server to Server RPC calls for Phoenix system tables
- **RpcServer.InvalidateMetadataCache.Fifo.handler** for metadata cache invalidation as part of redesign of Metadata caching

Each priority number used by Phoenix for above handler pools is unique and higher than 200. This is necessary because HBase recognizes 0 to 200 priority numbers and uses its own handler pools accordingly.

RpcSchedulerFactory provided by Phoenix:

- ClientRpcControllerFactory
- InterRegionServerIndexRpcControllerFactory
- InterRegionServerMetadataRpcControllerFactory
- InvalidateMetadataCacheControllerFactory

As of today, both Phoenix clients and servers use the default RpcSchedulerFactory as ClientRpcControllerFactory. Internally, ClientRpcControllerFactory provides MetadataRpcController. MetadataRpcController sets priority value corresponding to RpcServer.Metadata.Fifo.handler thread-pool for any HTable operation to be performed on

any of the Phoenix system tables. This results in RpcServer.Metadata.Fifo.handler being used for System table related RPC calls regardless of whether Phoenix client initiates the operation or Phoenix server.

Sample Problem Statement:

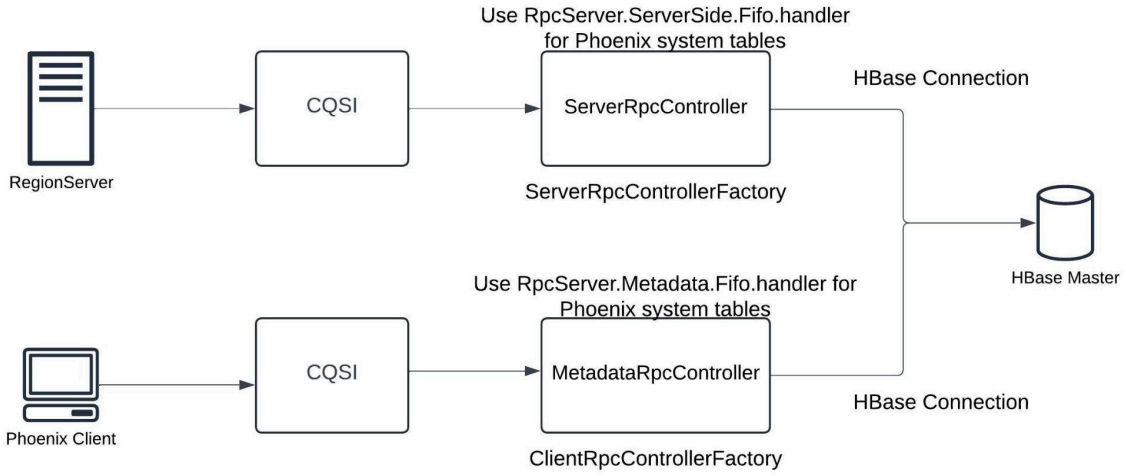
Phoenix client creates CQSI connection (`ConnectionQueryServices`), which maintains long time TCP connection with HBase server, usually known as `HConnection` or HBase Connection. Once the connection is created, it is cached by the Phoenix client.

While PHOENIX-6066 is considered the correct fix to improve the query performance, releasing it has surfaced other issues related to the RPC framework. One of the issues surfaced caused deadlock for SYSTEM.CATALOG serving regionserver as it could not make any more progress because all handler threads serving RPC calls for Phoenix system tables (thread pool: `RpcServer.Metadata.Fifo.handler`) got exhausted while creating server side connection from the given regionserver.
Several workflows from MetaDataEndpointImpl coproc require Phoenix connection, which is usually CQSI connection. Phoenix differentiates CQSI connections initiated by clients and servers by using a property: `IS_SERVER_CONNECTION`.
For CQSI connections created by servers, IS_SERVER_CONNECTION is kept true.
Under heavy load, when several clients execute getTable() calls for the same base table simultaneously, MetaDataEndpointImpl coproc attempts to create server side CQSI connection initially. As CQSI initialization also depends on Phoenix system tables existence check as well as client to server version compatibility checks, it also performs MetaDataEndpointImpl#getVersion() RPC call which is meant to be served by RpcServer.Metadata.Fifo.handler thread-pool. However, under heavy load, the thread-pool can be completely occupied if all getTable() calls try to initiate CQSI connection, whereas only a single thread can take global CQSI lock to initiate HBase Connection before caching CQSI connection for other threads to use. This has the potential to create deadlock.

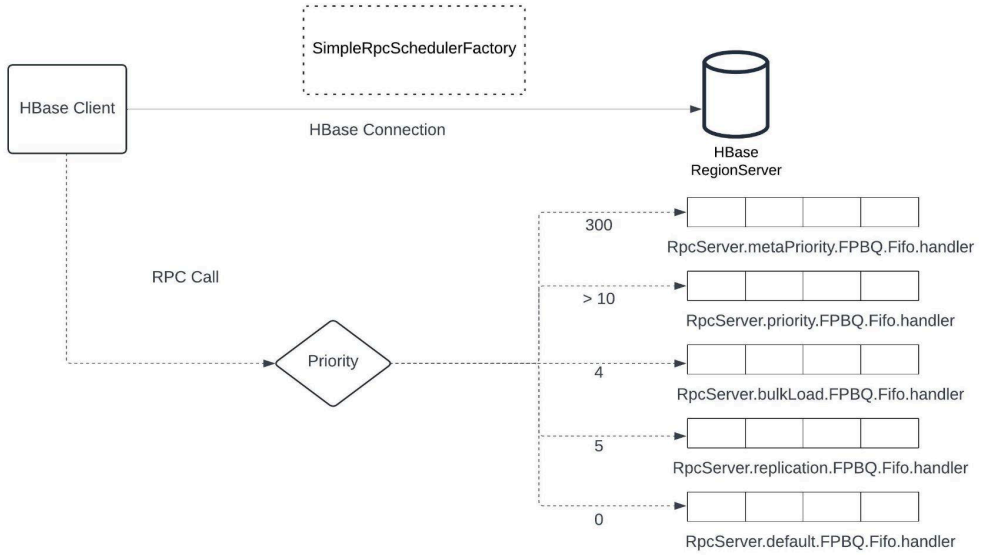
While the above mentioned problem statement is a specific case, there is a generic need for having different RPC handler pool for Phoenix system table RPC calls that are initiated by coprocessors (regionservers) than RPC handler pool used for Phoenix system table RPC calls that are initiated by clients.

Solutions:

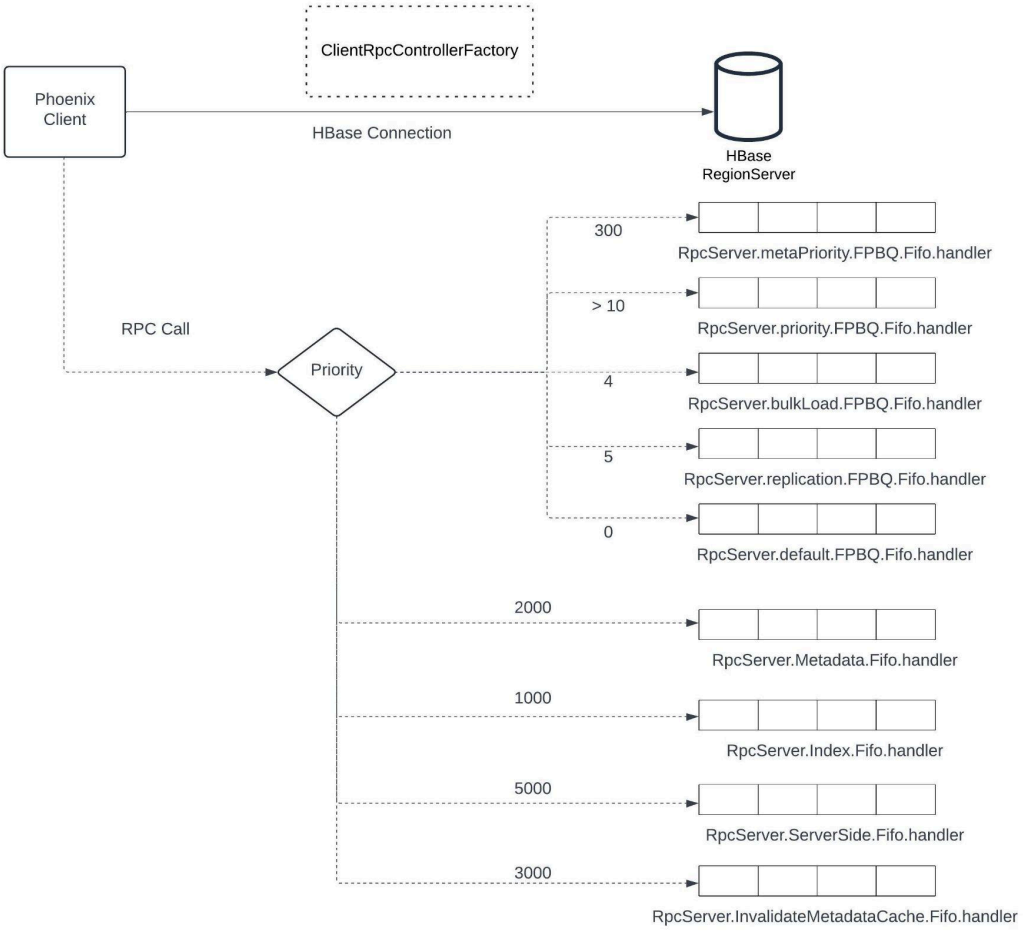
- Phoenix server to server system table RPC calls are supposed to be using separate handler thread-pools (PHOENIX-6687). However, this is not correctly working because regardless of whether the HBase Connection is initiated by client or server, Phoenix only provides ClientRpcControllerFactory by default. We need to provide a separate RpcControllerFactory during HBase Connection initialization done by Coprocessors that operate on regionservers.
- For the Phoenix server creating a CQSI connection, we do not need to check for existence of system tables as well as client-server version compatibility. This redundant RPC call can be avoided.



HBase Rpc Call priority function:



Phoenix Rpc Call priority function:


ClientRpcControllerFactory
Phoenix Client
HBase Connection
HBase RegionServer
300
RpcServer.metaPriority.FPBQ.Fifo.handler
> 10
RpcServer.priority.FPBQ.Fifo.handler
RPC Call
Priority
4
RpcServer.bulkLoad.FPBQ.Fifo.handler
5
RpcServer.replication.FPBQ.Fifo.handler
0
RpcServer.default.FPBQ.Fifo.handler
2000
RpcServer.Metadata.Fifo.handler
1000
RpcServer.Index.Fifo.handler
5000
RpcServer.ServerSide.Fifo.handler
3000
RpcServer.InvalidateMetadataCache.Fifo.handler