HBASE-9905: Enable using seqId as timestamp

Andrey Stepachev (octo47@gmail.com)

March 02, 2015

Timestamps

HBase timestamps.

Timestamps currently can be assigned by region server. This technique produce problems with consistency and even loss of edits become possible. A couple of situations why that can happen.

- time can goes backward and new inputs will have time in the past effectively put them before older edits
- region can be moved across region servers with different clocks, if new server is quite far behind from previous server, new edits will be not visible, they will be hidden by recent updates produced by server with clock in the future.
- client can experience situation when it will not see its own inserts querying data from different region servers, those servers will have different time, so client can't get consistent view to some point in time.

Clients can use their own source of time to set timestamps, but server still use wall clock to do compactions or cell version conflict resolution and deletions.

Proposing sequence/timestamp generators.

Basicly idea is to prefer client side timestamp generation or provide strict non-decrementing timestamp or sequence id generations on server. For client side that is relatively simple, we

just need to update timestamp with client time fixed at the start of request and optionally provide interface to make it possible for clients to substitute with their own time source.
For server side generation that can be a bit more complicated. But the main property of such generator should be non-decreasing.
To achieve this we can guarantee that timestamp never decreases for region. For timestamps that can be handled by ensuring that new timestamp is max(currentMillis, lastTimestamp+1).
For sequence id each region has associated atomic counter which is initialized with maxSeq + 1 on region load.
For further optimisations each generator can promise monotony of its sequence.
Server side sequences due of planned implementation will be always monotony within one region. Monotony can be per region or global. Global is out of scope now, it complicates design for now, so we will go with per-region only.

TTL

With timestamps it is easy to define ttl() function. With sequence it is not obvious how to calculate TTL for every cell. For server side sequences support for cell TTLs can be disabled, only max number of versions can be supported. For client side sequences table can have MIN_SEQ (or MIN_TIMESTAMP for name coherency) for each table in META and it will be used during compaction to determine cell eligibility for removal. Moreover, this value can be used for temporary override minimal value for TTL calculation for any table. If such value is set for any particular table it overrides normal ttl calculations and this value used for making decision of the cell removal.

Split/Merge

Split regions will share the same sequence at the start, but later can diverge. For merges maximum of sequences should be taken.

That gives following table of possible generators:

Name	Client/Server Side	Semantics	Possible properties
client_ts	client	timestamp	monotone
client_seq	client	sequence	monotone
server_ts	server	timestamp	monotone
server_seq	server	sequence	always monotone

Implementation.

Making generator type persistent.

Information about current sequence generator need to be kept somewhere. We have two possible place where it can be stored persistently: meta, table descriptor or regioninfo.

Meta

Meta is a good place to keep it, but it became impossible to access this information if meta or cluster is offline. That make it not possible to correctly write data with bulk load or recover data or check region compatibility (generally it is not a good idea to combine different generators in one region).

Table Descriptor

Table generator can be kept in table descriptor. In that case this information can be accessible at any time even offline. But that doesn't scale well, getHTableDescriptor methods will hit master on every new HTable instance created. That is because HTable need to know what generator to use. We can cache that, but every new client still will hit master, and if master is not accessible, we will be unable to create connections.

HRegionInfo

Type of generator can be kept in each region. Somehow that gives flexibility (different regions can have different generators, if that suits user). Generator compatibility can be validated by HBaseFsck without accessing master or meta. For split it is straightforward to find out derived generator type, for merges that will not work. Type of generator can be piggybacked into region locations and be kept in meta or served directly by region servers.

Conclusion

Looks like combined solution can be used. Primary type of generator can be stored in HTableDescriptor. That make it persistent and easily accessible by offline tools. Each new region will get current generator from HTableDescriptor. Altering of generator can be not supported for multi-region tables in first release, but later can be supported via enable/disable table (during region onlining generator can be checked and updated).
Basically that gives one source of generator type, but keeping master off write pass, region servers will respond for answering type of generator used by table (per region to be more precise).

Instantiating

Generator should be configured in hbase-site by properties hbase.sequence.generator.(client|server).TYPE.class and placed into registry.
We need to have separate implementations of generators for client and server side. For example client side should generate timestamp for client_ts but server side should just check that timestamp actually was set

Client

Registry kept in ConnectionManager and shared across all connections. Connection using information from RegionLocation allows to know which generator should be used. AsyncProcess will use registry of generators to enhance mutations with proper timestamps/sequenceIds.

Server

Registry will be kept per region server (accessible through RegionServerServices). Each region on load will instantiate generator according to last sequence known by region.