Counters

Counters in Cassandra

Getting started

In order to use counters in Cassandra you need to first set up a cluster, see the Getting started guide for more information. Also note that counters are available starting with Cassandra 0.8.0.

Configuration

To use counters, you have to define a column family (or super column family) whose columns will act as counters. To create such a column family `counterCF` using the CLI, you will have to do:

```
[default@test] create column family counterCF with default_validation_class=CounterColumnType and
replicate_on_write=true;
```

Setting the `default_validation_class` to `CounterColumnType` indicates that the columns will be counters. Setting `replicate_on_write=true` will be optional starting in 0.8.2, but a bug made it default to false in 0.8.0 and 0.8.1, which is unsafe.

Incrementing and accessing counters

Once the cluster is up and running with these settings you can simply increment or decrement long values from the counters as follows (example is using the python thrift client):

```
client.add('key1', ColumnParent(column_family='Counter1'), CounterColumn('c1', 100), ConsistencyLevel.ONE)
client.add('key1', ColumnParent(column_family='Counter1'), CounterColumn('c1', -50), ConsistencyLevel.ONE)
```

And read it back

```
rv = client.get('key1', ColumnPath(column_family='Counter1', column='c1'), ConsistencyLevel.ONE).counter_column.
value
```

Please read the rest of this wiki page, especially Technical limitations and Operational considerations to make sure this actually does what you need.

Interface

The interface follows the main API. The main differences are:

- CounterColumn requires an i64 value (can be negative) and no timestamp,
- Counter can be used inside super columns using the CounterSuperColumn structure, and
- Deletion, when used on a counter column family, does not use a timestamp.

Internally, the data store generates timestamps on the server to determine priority of deletion.

The new structures for dealing with counters are:

```
struct CounterColumn {
    1: required binary name,
    2: required i64 value
}

struct CounterSuperColumn {
    1: required binary name,
    2: required list<CounterColumn> columns
}

struct ColumnOrSuperColumn {
    1: optional Column column,
    2: optional SuperColumn super_column,
    3: optional CounterColumn counter_column,
    4: optional CounterSuperColumn counter_super_column
}
```

where the pre-existing ColumnOrSuperColumn has the two new fields, specific to counters, `counter_column` and `counter_super_column`.

Moreover, as mentioned previously, the timestamp field of Deletion is now optional (but remain mandatory for non counter column family operation).

The counter operations comprise the usual `batch_mutate`, `get`, `get_slice`, `multiget_slice`, `multiget_count` and `get_range_slice` (secondary indexes on counter column family is not supported at the moment), as well as the following new operations for access to a single counter:

```
# counter methods

/**
 * Increment or decrement a counter.
 */
void add(1:required binary key,
         2:required ColumnParent column_parent,
         3:required CounterColumn column,
         4:required ConsistencyLevel consistency_level=ConsistencyLevel.ONE)
     throws (1:InvalidRequestException ire, 2:UnavailableException ue, 3:TimedOutException te),

/**
 * Remove a counter at the specified location.
 */
void remove_counter(1:required binary key,
                    2:required ColumnPath path,
                    3:required ConsistencyLevel consistency_level=ConsistencyLevel.ONE)
    throws (1:InvalidRequestException ire, 2:UnavailableException ue, 3:TimedOutException te),
```

Technical limitations

- If a write fails unexpectedly (timeout or loss of connection to the coordinator node) the client will not know if the operation has been performed. A retry can result in an over count CASSANDRA-2495.
- Counter removal is intrinsically limited. For instance, if you issue very quickly the sequence "increment, remove, increment" it is possible for the removal to be lost (if for some reason the remove happens to be the last received messages). Hence, removal of counters is provided for definitive removal only, that is when the deleted counter is not increment afterwards. This holds for row deletion too: if you delete a row of counters, incrementing any counter in that row (that existed before the deletion) will result in an undetermined behavior. Note that if you need to reset a counter, one option (that is unfortunately not concurrent safe) could be to read its *value* and add *-value*.
- `CounterColumnType` may only be set in the `default_validation_class`. A column family either contains only counters, or no counters at all.

Further reading

See CASSANDRA-1072 and especially the design doc for further information about how this works internally (but note that some of the limitations fixed in these technical documents have been fixed since then, for instance all consistency level **are** supported, for both reads and writes).

`https://c.statcounter.com/9397521/0/fe557aad/1/|stats`