Paper - Google Bigtable

TEAM	
Akshay Aurora	2011B4A7658P
Atul Mittal	2011B4A7861P
Siddharth Bhatia	2011B4A7680P

Setup

- A single node runs master server - responsible for operations on tables
- Each datanode in cluster runs tablet server - responsible for serving table rows
- Both master and tablet server handle requests concurrently using threads
- HDFS replication factor for the hadoop cluster (5 - 6 nodes) will be set to 2-3
- Client can be run outside the cluster

Implementation Choice

- HDFS (Java API)
- Communication b/w client and cluster nodes using TCP sockets
- JSON is used for serialization and deserialization of messages and storage of data in files

Client Operations

- **Client creates table with column families**
    - client sends create request to master
    - master creates table file on HDFS
- **Client opens table**
    - client sends open table request to master
    - master reads table contents from HDFS
    - returns clients with table data
- **Client adds row in bigtable**
    - column families of the rows are validated with those in table.
    - client sends add row request with row data to tablet server
- **Client reads row from bigtable**
    - client sends read row request to tablet server
    - tablet server searches memtable, if key is not found files on disk are searched
    - return client with row data

- **Client updates a row in bigtable**
    - column families of the rows are validated with those in table.
    - client sends update row request to tablet server
    - tablet server updates row in memtable
- **Client deletes a row in bigtable**
    - client sends delete row request to tablet server
    - tablet server removes row from memtable
- **Client deletes table**
    - Send delete request to master
    - master deletes table file from HDFS

Storage

- **Storage in tablet server**
    - Each tablet server has in memory table (memtable) serving serving rows from different tables
    - Memtables work as long as we can fit all the data in memory. We demonstrate this feature on the number of records.
        - When the number of entries in memtable exceeds certain number the table is written to HDFS.
    - Each Tablet server stores files in its own directory on HDFS - $HOME/*NodeName*
    - Data for each tablet in single file (*TableName@FileNumber*.tablet)
        - Where *FileNumber* tells how recently the file was created
    - Tablets file contain table records in JSON format.
    - The tablets files on disk are immutable like SSTables in BigTable
    - To find a particular key, we check the key in memtable and then tables in reverse chronological order using the first value.
- **Storage in master server**
    - Master server handles requests for creating/updating/opening/deleting tables
    - Master stores files in its own directory on HDFS - $HOME/master
    - For each table master creates a file - *TableName*.smalltable

- Table Format (*TableName*.smalltable)
    -

```
{
      tableName: webtable,
      families: ["lang", "content", "anchor"],
      tablets: {
            "A, Z" : "tabletserver1 : port",
            "a, z" : "tabletserver2 : port"
      }
  }
```

- Table Map Format (T*ableName@FileNumber*.tablet)
    -

```
{
      "key1": {
            "family1": {
                  "field": {
                        5 : "value3",
                  },
            "family2": {
                  "field2": {
                        19 : "value1",
                        10 : "value2",
                        5  : "value3",
                  }
            }
      },
      "key2": {
            "family1": {
                  "field": {
                        9 : "value1",
                        5 : "value3",
                  },
            "family2": {
                  "field2": {
                        19 : "value1",
                        10 : "value2",
                  }
            }
      }
}
```

Client API

- Table operations
  - Create table object
    - SmallTable table = new SmallTable("simpletable");
  - Adding column families to table
    - table.addColumnFamily("family1");
  - Create table
    - table.create();
  - Open table
    - table.open()
  - Update table
    - table.update()
- Table Row operations
  - Add row to table
    - table.addRow(row);
  - Get row from table
    - table.getRow(key);
  - Update row from table
    - table.updateRow(key, row);
  - Delete row from table
    - table.deleteRow(key);
- Row operations
  - Create row object
    - SmallRow row = new SmallRow();
  - Add column value with timestamp
    - row.setColumn("familyname:foo", "hello", 5);
  - Add column value
    - row.setColumn("familyname:foo", "hello");
  - Get column value with latest timestamp
    - row.getColumn("familyname:foo");
  - Get column value with specific timestamp value
    - row.getColumn("familyname:foo", 5);
  - Get value with latest timestamp
    - row.getValue("familyname:foo");
  - Get value with specific timestamp
    - row.getValue("familyname:foo", 5)
  - Get column family value
    - row.getFamily("familyname");