HBase – Secondary Index

Anoop Sam John

www.huawei.com

HUAWEI

Huawei Hadoop R&D – In a Glance

Hadoop Development

- Secondary Index in HBase
- HDFS NN HA (Hadoop-2)
- Bookkeeper as shared storage for NN HA (Hadoop-2)
- HDFS NN HA (Hadoop-1)
- MapReduce ResourceManager HA (Hadoop-2 / YARN)
- MapReduce JobTracker HA (Hadoop-1)
- Hive HA

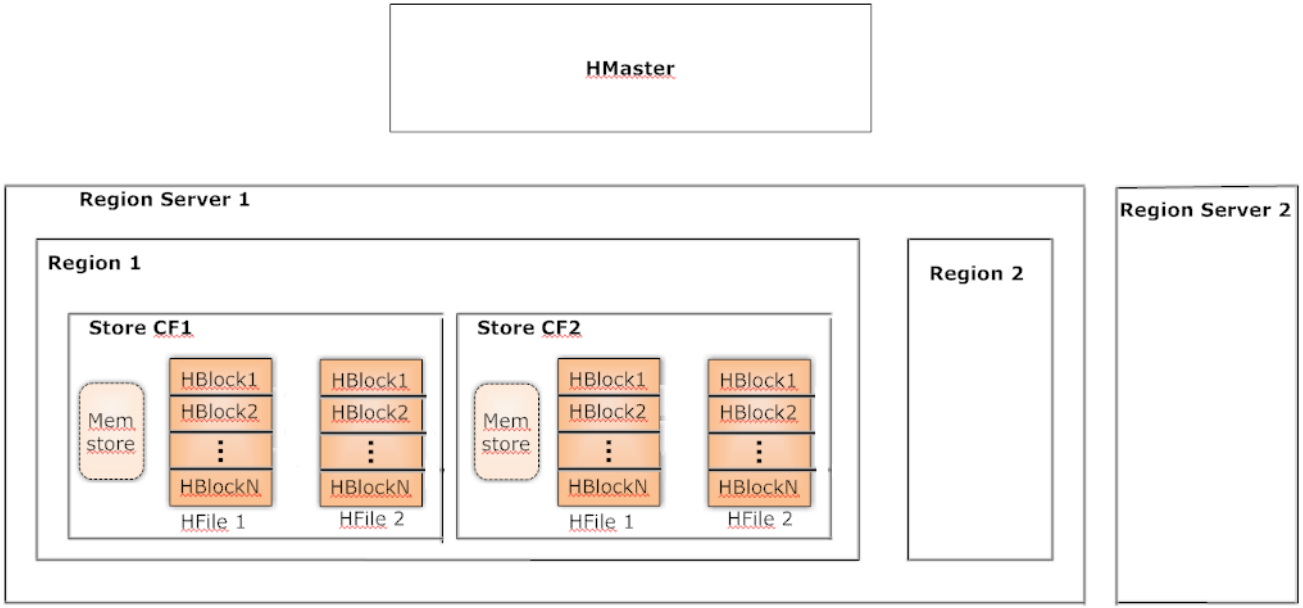
Stabilization

- Raised over 650 defects since Jan'11
- Fixed over 500 defects since Jan'11, and contributed back to community.

Who am I

- Senior Tech Lead in Huawei R&D centre @Bangalore
- Active contributor in Apache HBase community
- Active member in HBase dev/user mailing list
- Working with HBase and Hive

HBase Recap

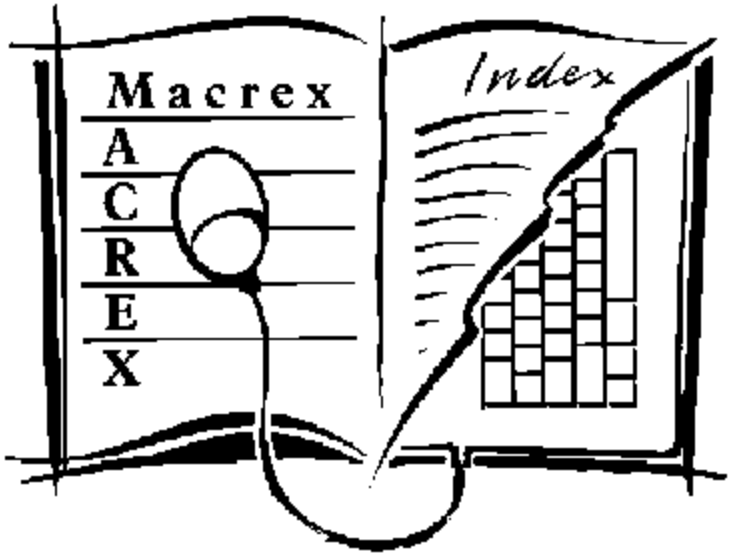


- Master, Region servers
- Table split into regions
- Columnar storage, Column family
- Memstore, Hfiles in DFS
- Hfiles logically split into smaller blocks, data write/read as blocks

Motivation

- No indexing available in HBase
- Scans with conditions on column value on huge sparse data.
- Filter usage
  - Scan throughput very less
  - Timeouts at client side and

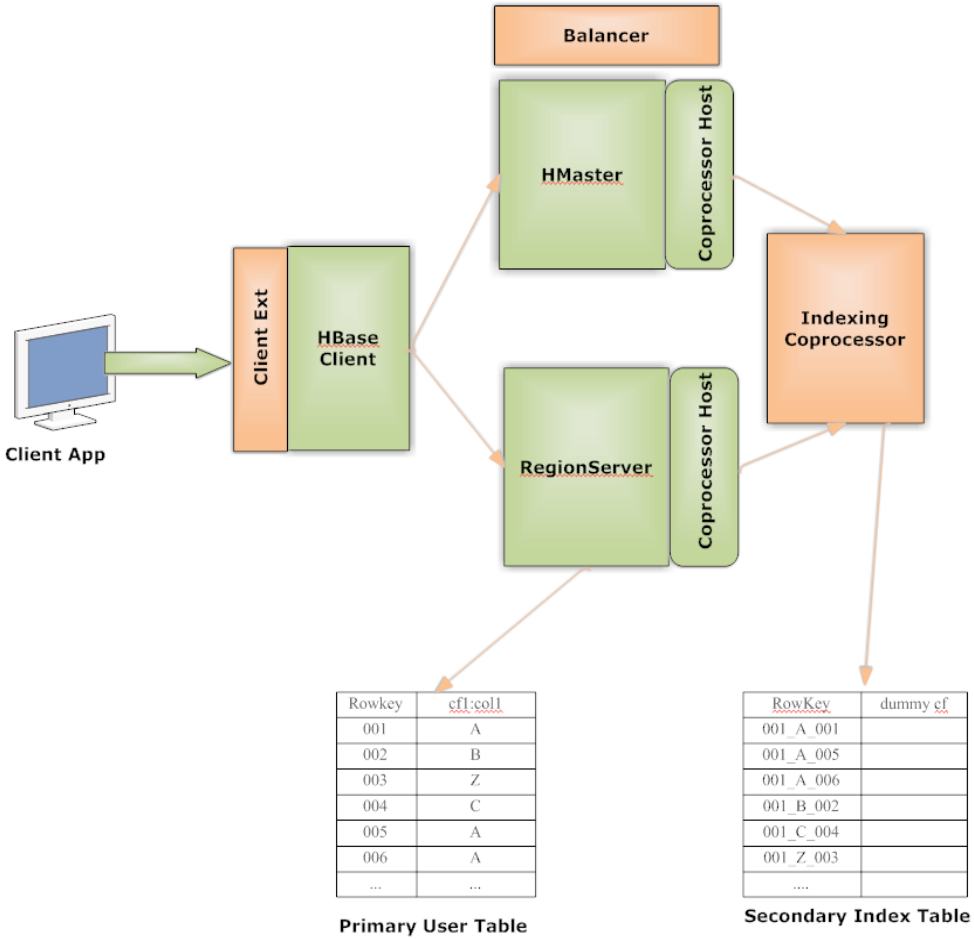
    lease expiry



Overall Solution

- Index table
  - Persist index data in separate table
- Co-processor based design
  - 100% server side solution
- Region wise indexing with collocation
  - 1-1 mapping with actual table and index table regions
  - Region collocation using custom Load balancer

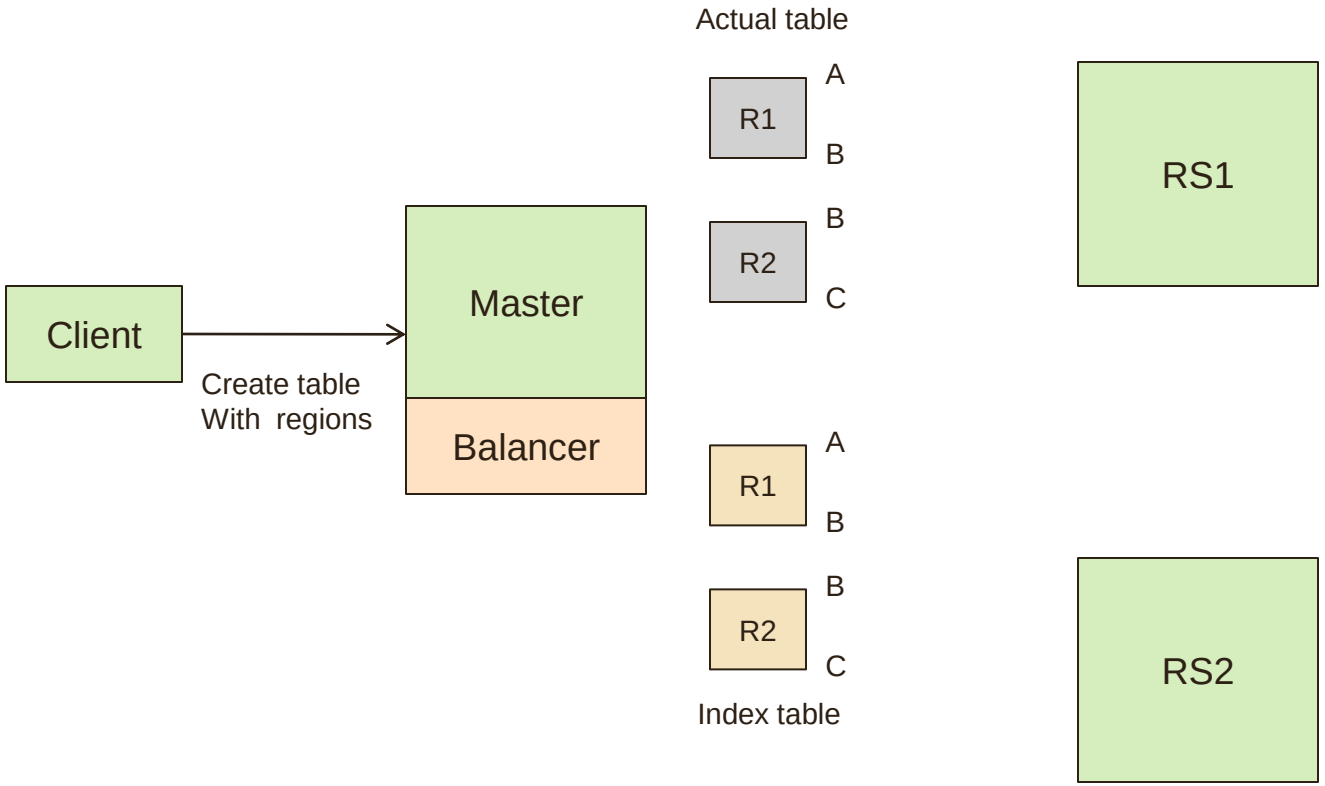
Secondary Index Architecture



- Client Ext allows to specify index details while table create
- Custom balancer do collocation
- Coprocessor handles the index data

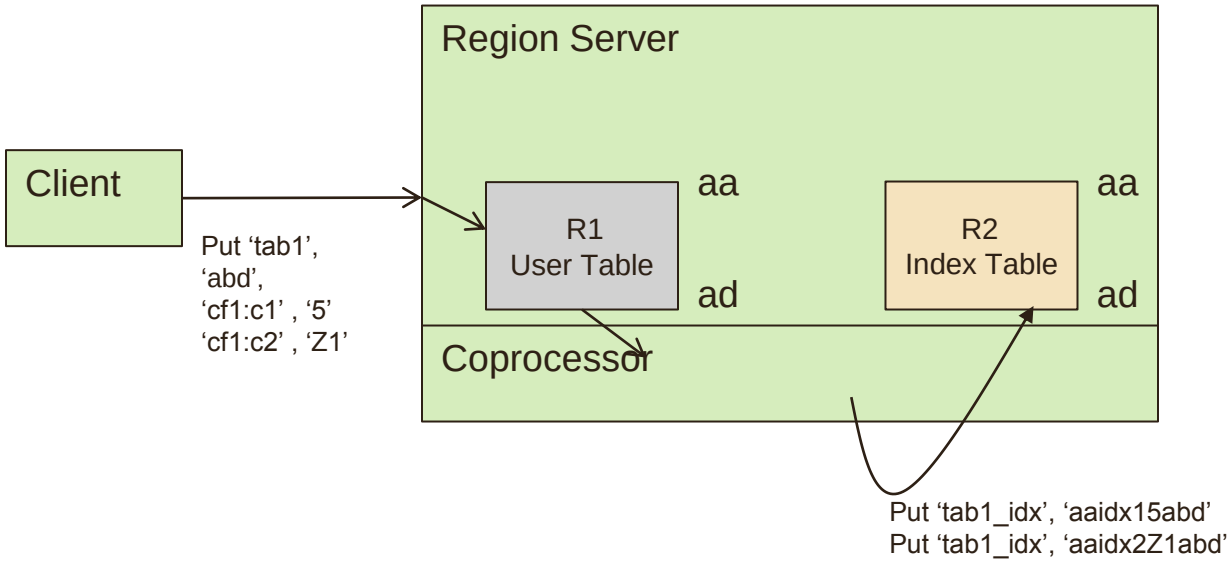
Table creation

Regions collocation



Put operation

- Table –> tab1 column family –> cf1
- Index –> idx1, cf1:c1 and idx2, cf1:c2
- Index table –> tab1_idx

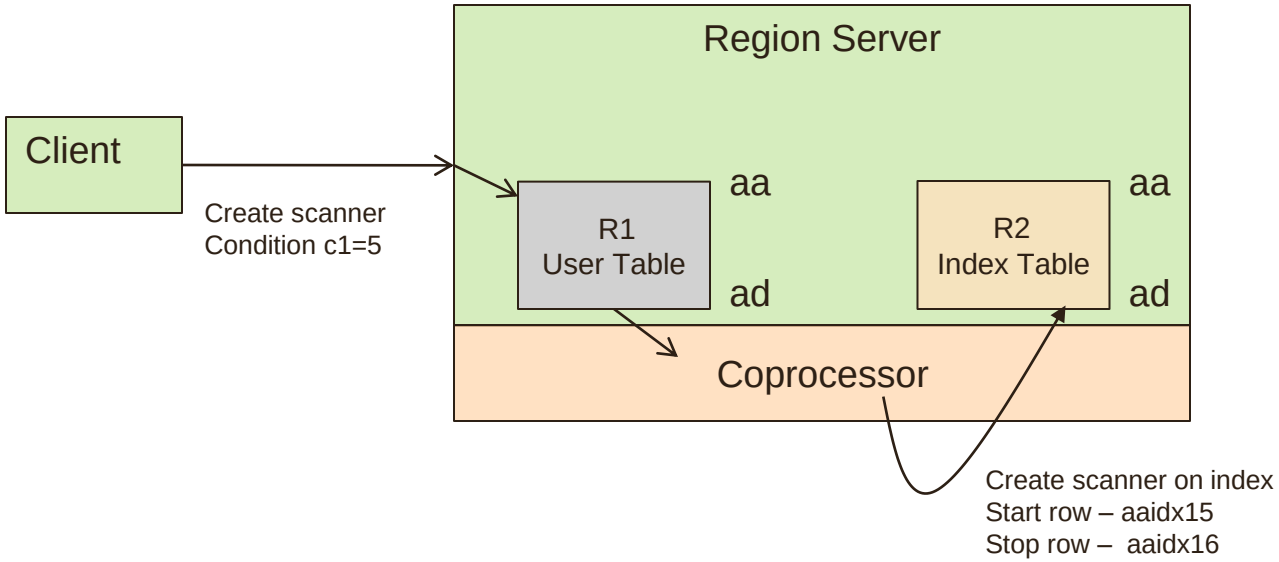


Index table rowkey

= region startkey + index name + indexed column value + user table rowkey

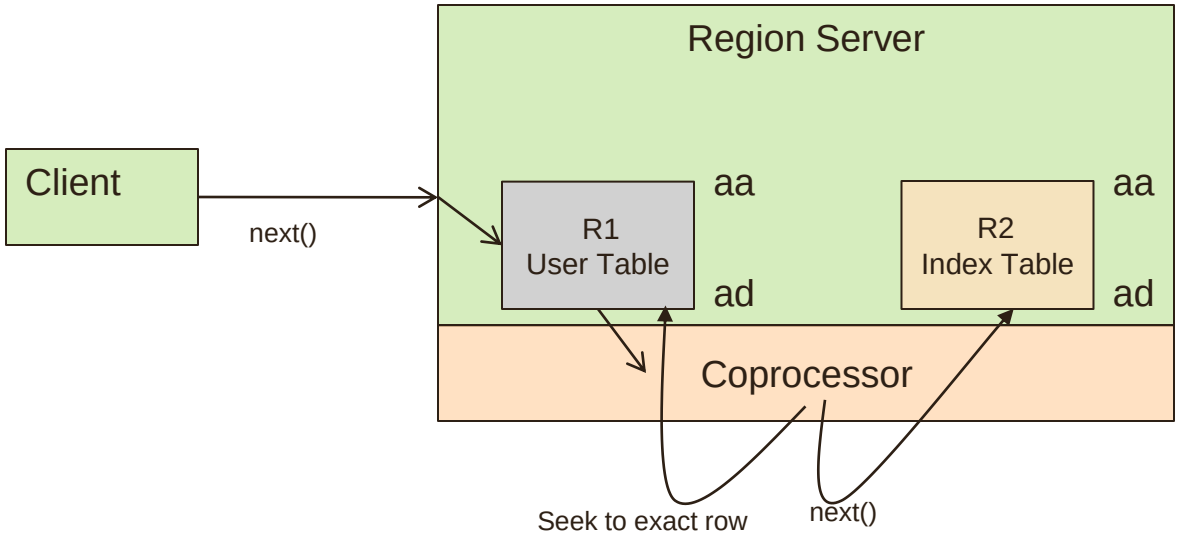
- One index table per user table
- All index data goes to same index table

Scan operation



- Creating Scanner for index table at sever side

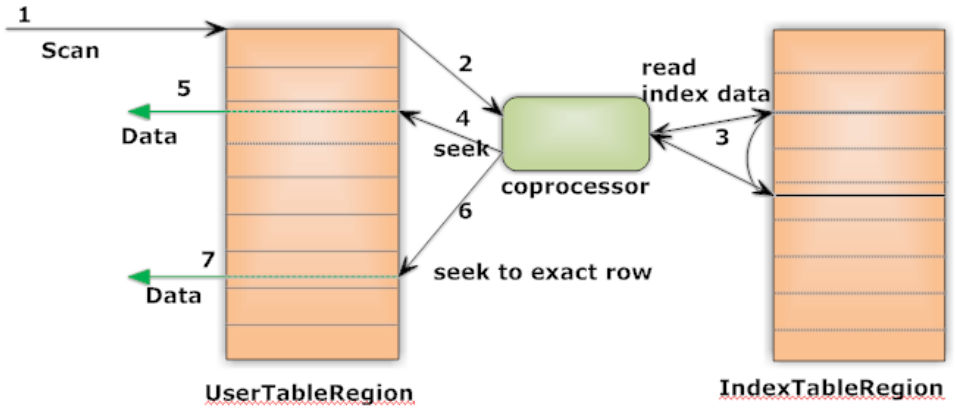
Scan operation

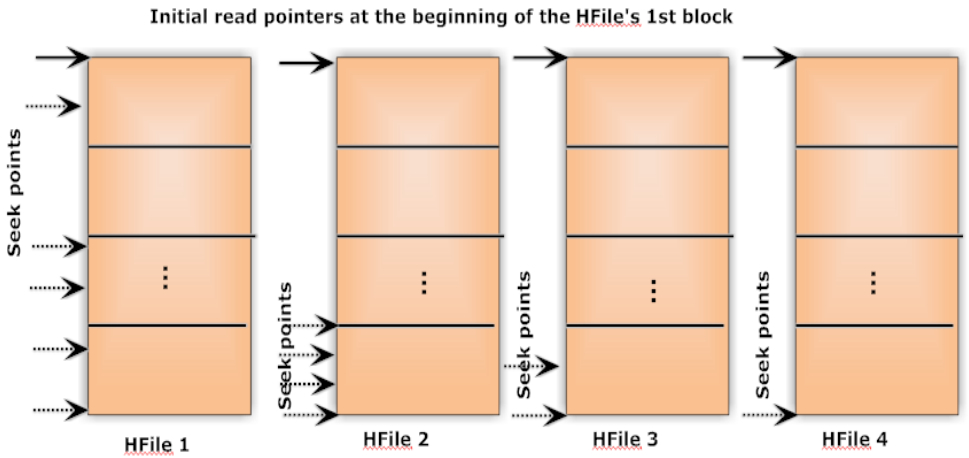


- Scan index data and seek to exact rows in the user table
- Index usage at server side

Scan operation

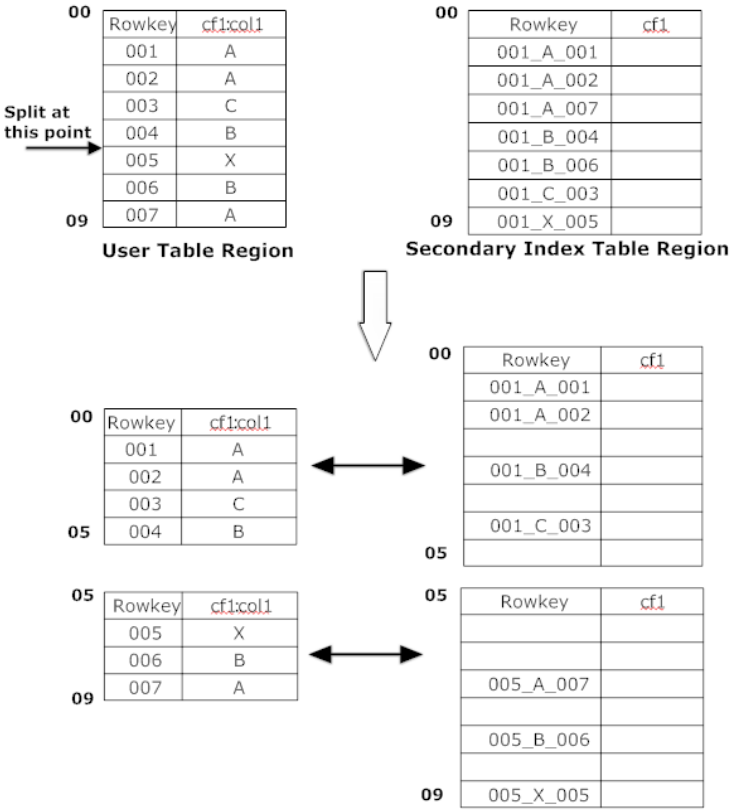
- CP reads index data and seek to exact rows in user table
- Doing seeks on HFiles based on rowkey obtained from index data
- HFiles reads as block by block
  - Default block size is 64KB
- Skipping block reads from HDFS where data not at all present
- Some times skip a full HFile ☺
- No need to read index details back to client avoiding network extra usage

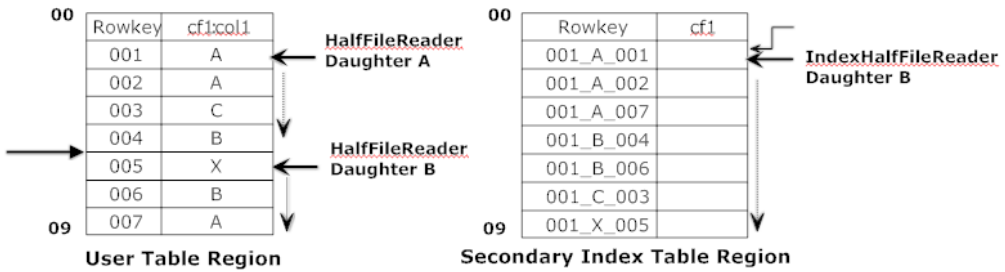




Region Split

- Explicit split on index region is avoided
- When user table region splits, corresponding index region also splits
  - Split key for index region same as that of user region
  - Custom HalfStoreFileReader for index daughter regions

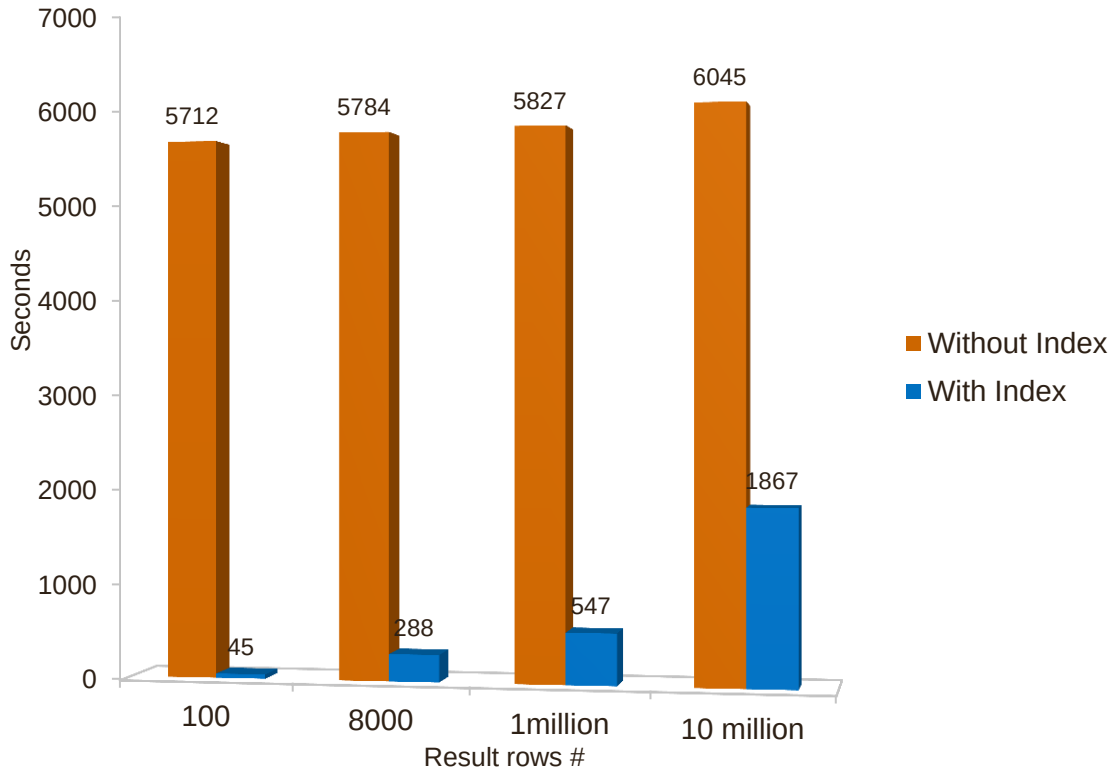




- IndexHalfStoreFileReader – Both half region readers starts at same point ie. begin of Hfile
- Checks the actual table rowkey part and decide KV corresponds to it or not
- IndexHalfStoreFileReader for daughter B – changes the key as per the daughter region startkey.

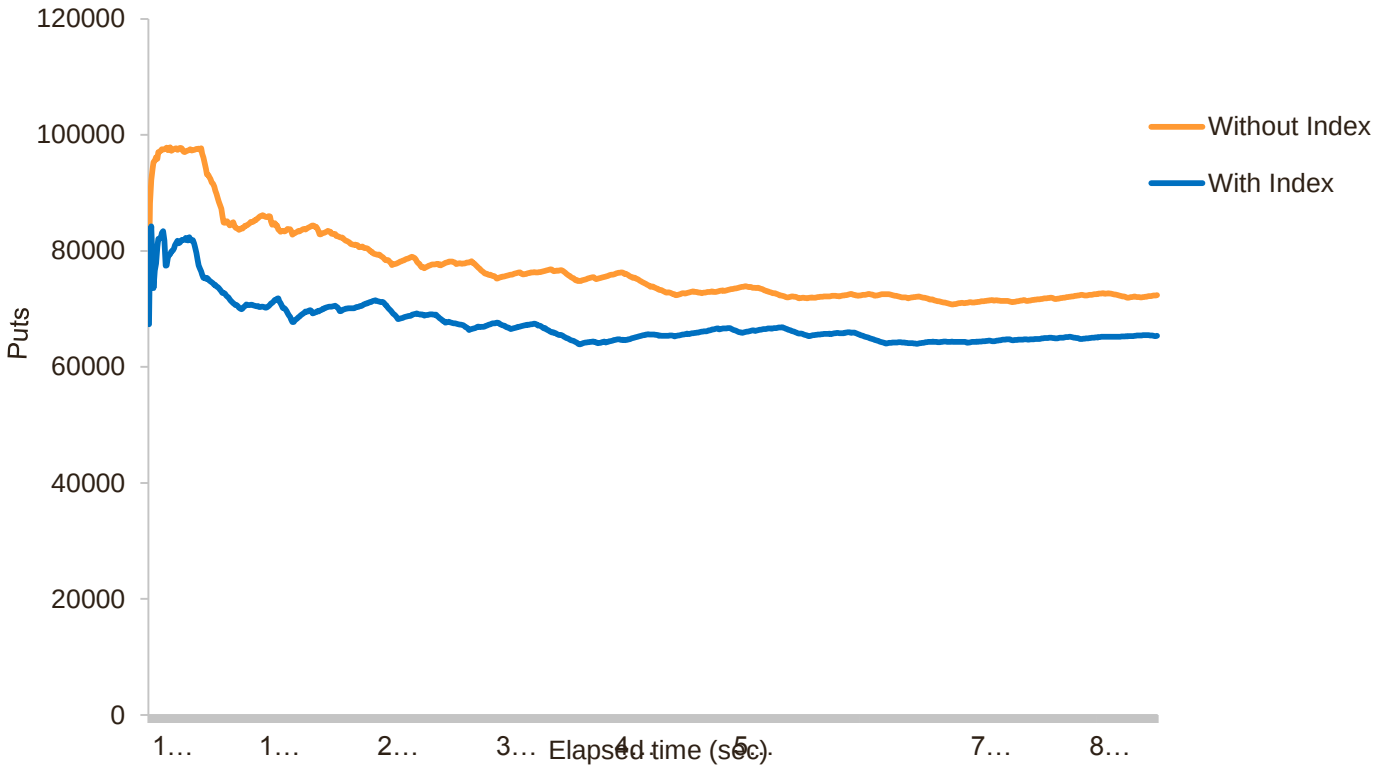
Performance Test Results (Scan)

- 4 Region Servers. Table regions # 160
- Total records : 570 million
- Per record size : 800 bytes
- EQUALS condition on a column



Performance Test Results (Put)

- 4 Region servers, table with 160 regions
- ~10% degradation in put performance



Salient aspects

- **Design**
  - Supports multiple index on table and multi column index
  - Support indexing part of a column value
  - Support for equals and range condition Scans using index.
  - Support to dynamically add/drop index
  - Support bulk loading data to indexed table – Indexing done with bulk load
- **Application Usage**
  - No change in scan code for the client app.
  - No need to specify the index(s) to be used. Intelligence to find best index(s) to be used for a Scan by looking at the Filters.
- **Upgrade / Integration**
  - Very less code changes in HBase core. HBase version upgradation is very easy for us

Q&A?

Thank you

www.huawei.com