# Ecosystems built with HBase and CloudTable service at Huawei

Jieshan Bi, Yanhui Zhong

# Agenda

**CTBase: A light weight HBase client for structured data**

Tagram: Distributed bitmap index implementation with HBase

CloudTable service(HBase on Huawei Cloud)

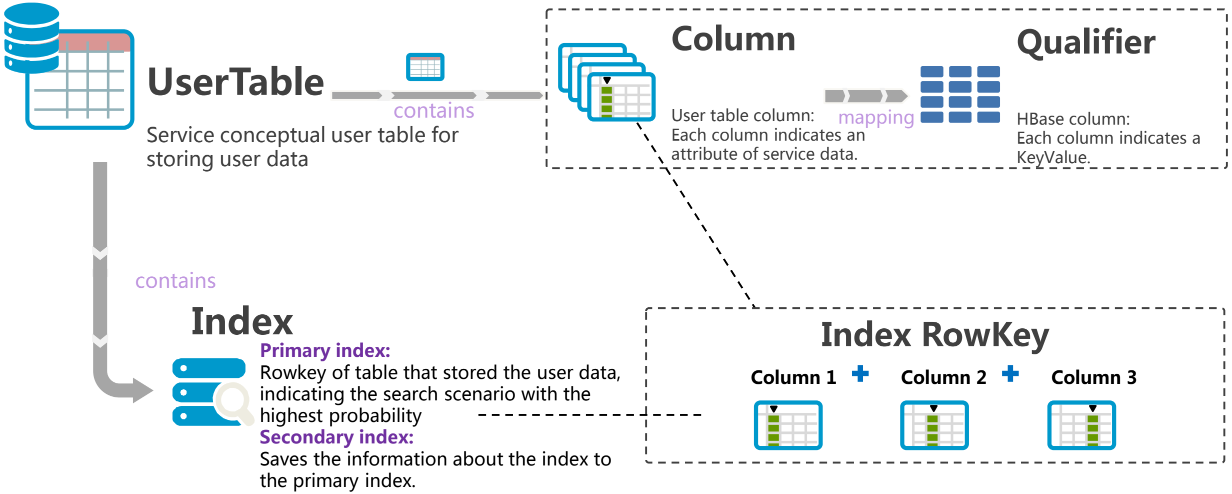
# CTBase Design Motivation

- Most of our customer scenarios are structured data
- HBase secondary index is a basic requirement
- New application indicated new HBase secondary development
- Simple cross-table join queries are common
- Full text index is also required for some customer scenarios

# CTBase Features

- Schematized table
- Global secondary index
- Cluster table for simple cross-table join queries
- Online schema changes
- JSON based query DSL

# Schematized Table


UserTable
contains
Service conceptual user table for storing user data
Column
Qualifier
User table column:
Each column indicates an attribute of service data.
mapping
HBase column:
Each column indicates a KeyValue.
contains
Index
Primary index:
Rowkey of table that stored the user data, indicating the search scenario with the highest probability
Secondary index:
Saves the information about the index to the primary index.
Index RowKey
Column 1
Column 2
Column 3

Schematized Tables is better for structured user data storage. A lot of modern NewSQL databases likes MegaStore, Spanner, F1, Kudu are designed based on schematized tables.

# Schema Manager

CTBase provide schema definition API. Schema definition includes:

- **Table Creation**

A user table will be exist as simple or cluster table mode.

- **Column Definition**

Column is a similar concept with RDBMS. A column has specific type and length limit.

- **Qualifier Definition**

Column to ColumnFamily:Qualifier mapping. CTBase supports composite column, multiple column can be stored into one same ColumnFamily:Qualifier.

- **Index Definition**

An index is either primary or secondary. The major part of index definition is the index rowkey definition. Some hot columns can also be stored in secondary row.

# Schema Manager Cont.

- **Meta Cache**

Each client has a schema locally in memory for fast data conversion.

- **Meta Backup/Recovery Tool**

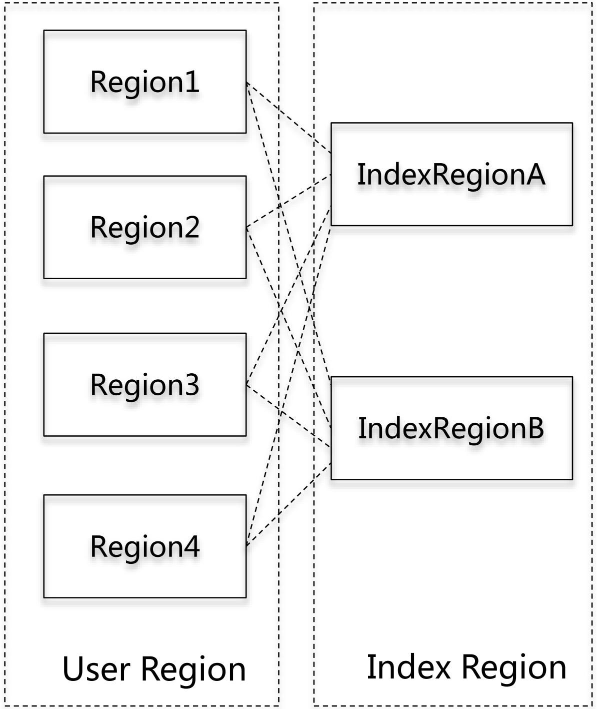
Schema data can be exported as data file for fast recovery.

- **Schema Changes**

- Column changes
- Qualifier changes
- Index changes

Some changes are light-weight since they can take advantage of the scheme-less characteristics of HBase. But some changes may cause the existing data to rebuild.

# HBase Global Secondary Index



NAME = 'Lina'

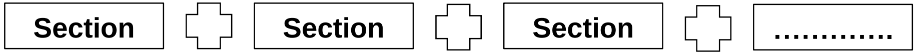
| NAME | ID |
|---|---|
| Ariya | I0000005 |
| Bai | I0000006 |
| He | I0000004 |
| Lily | I0000001 |
| **Lina** | **I0000003** |
| **Lina** | **I9999999** |
| Lisa | I0000008 |
| Wang | I0000002 |
| Wang | I0000007 |
| ……. | …………. |
| Xiao | I0000009 |

| ID | NAME | PROVINCE | GENDER | PHONE | AGE |
|---|---|---|---|---|---|
| I0000001 | Lily | Shandong | MALE | 13322221111 | 20 |
| I0000002 | Wang | Guangdong | FEMAIL | 13222221111 | 15 |
| **I0000003** | **Lina** | **Shanxi** | **FEMAIL** | **13522221111** | **13** |
| I0000004 | He | Henan | MALE | 13333331111 | 18 |
| I0000005 | Ariya | Hebei | FEMAIL | 13344441111 | 28 |
| I0000006 | Bai | Hunan | MALE | 15822221111 | 30 |
| I0000007 | Wang | Hubei | FEMAIL | 15922221111 | 35 |
| I0000008 | Lisa | Heilongjiang | MALE | 15844448888 | 38 |
| I0000009 | Xiao | Jilin | MALE | 13802514000 | 38 |
| …………. | ……. | …… | ………. | ………………….. | …. |
| **I9999999** | **Lina** | **Liaoning** | **MALE** | **13955225522** | **70** |

Secondary index is for non-key column based queries.
Global secondary index is better for OLTP-like queries with small batch results.

# HBase Global Secondary Index Cont.

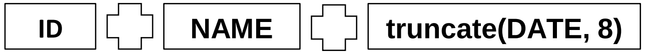
## Index RowKey Format



Section is normally related to one user column, but can also be a constant or a random number.

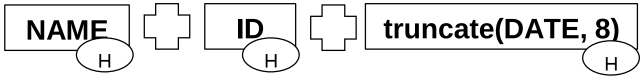
## Primary Key

Suppose table UserInfo includes below 5 columns：
ID, NAME, ADDRESS, PHONE,DATE
Primary key are composed with 3 sections：
Section 1: ID
Section 2: NAME
Section 3: truncate(DATE, 8)
So the primary rowkey is:

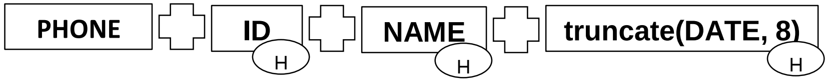


## Secondary Index Key

Secondary index key for NAME index:



Secondary index key for PHONE index:



NOTE：Sections with Ⓗ are also exist in primary key

# Cluster Table

**Example: select** a.account_id, a.amount, b.account_name, b.account_balance from **Transactions** a
**left join AccountInfo** b on a.account_id = b.account_id **where** a.account_id = "xxxxxxx"

| account_id | amount | time |
|---|---|---|
| A0001 | $100 | 12/12/2014 18:00:02 |
| A0001 | $1020 | 10/12/2014 15:30:05 |
| A0001 | $89 | 09/12/2014 13:00:07 |
| A0002 | $105 | 11/12/2014 20:15:00 |

| account_id | account_name | account_balance |
|---|---|---|
| A0001 | Andy | $100232 |
| A0002 | Lily | $902323 |
| A0003 | Selina | $90000 |
| A0004 | Anna | $102320 |

| | | |
|---|---|---|
| A0001 | Andy | $100232 |
| A0001 | $100 | 12/12/2014 18:00:02 |
| A0001 | $1020 | 10/12/2014 15:30:05 |
| A0001 | $89 | 09/12/2014 13:00:07 |
| A0002 | Lily | $902323 |
| A0002 | $105 | 11/12/2014 20:15:00 |
| A0002 | $129 | 11/11/2014 18:15:00 |

Records from different business-level user table stored together

AccountInfo record

Transaction record

**Pre-Joining with Keys: A better solution for cross-table join in HBase**. Records come from different tables but have some same primary key columns can be stored adjacent to each other, so the cross-table join turns into a sequential scan.

# ClusterTable Write Vs. HBase Write

```
Table table = null;
try {
    table = conn.getTable(TABLE_NAME);

    // Generate RowKey.
    String rowKey = record.getId() + SEPERATOR + record.getName();

    Put put = new Put(Bytes.toBytes(rowKey));
    // Add name.
    put.add(FAMILY, Bytes.toBytes("N"), Bytes.toBytes(record.getName()));
    // Add phone.
    put.add(FAMILY, Bytes.toBytes("P"), Bytes.toBytes(record.getPhone()));
    // Add composite columns.
    String compositeColumn = record.getAddress() + SEPERATOR
      + record.getAge() + SEPERATOR + record.getGender();
    put.add(FAMILY, Bytes.toBytes("Z"), Bytes.toBytes(compositeColumn));

    table.put(put);
  } catch (IOException e) {
    // Handle exception.
  } finally {
    // ……..
}
```

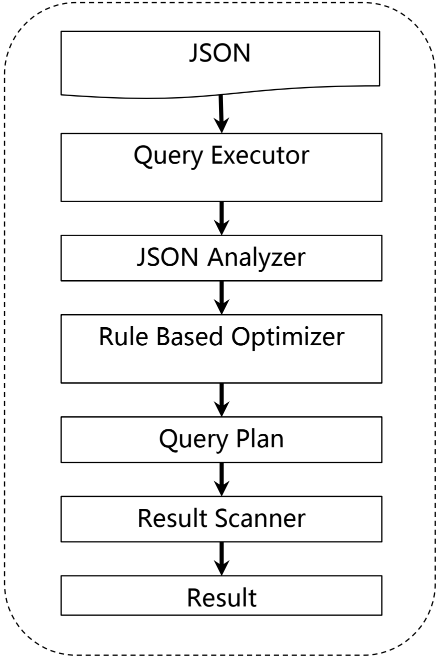
```
ClusterTableInterface table = null;
try {
    table = new ClusterTable(conf, CLUSTER_TABLE);

    CTRow row = new CTRow();

    // Add all columns.
    row.addColumn("ID", record.getId());
    row.addColumn("NAME", record.getName());
    row.addColumn("Address", record.getAddress());
    row.addColumn("Phone", record.getPhone());
    row.addColumn("Age", record.getAge());
    row.addColumn("Gender", record.getGender());
    table.put(USER_TABLE, row);
  } catch (IOException e) {
    // Handle exception.
  } finally {
    // ………….
 }
```

RowKey/Put/KeyValue are not visible to application directly. Secondary index row will be auto-generated by CTBase.

# JSON Based Query DSL



JSON

Query Executor

JSON Analyzer

Rule Based Optimizer

Query Plan

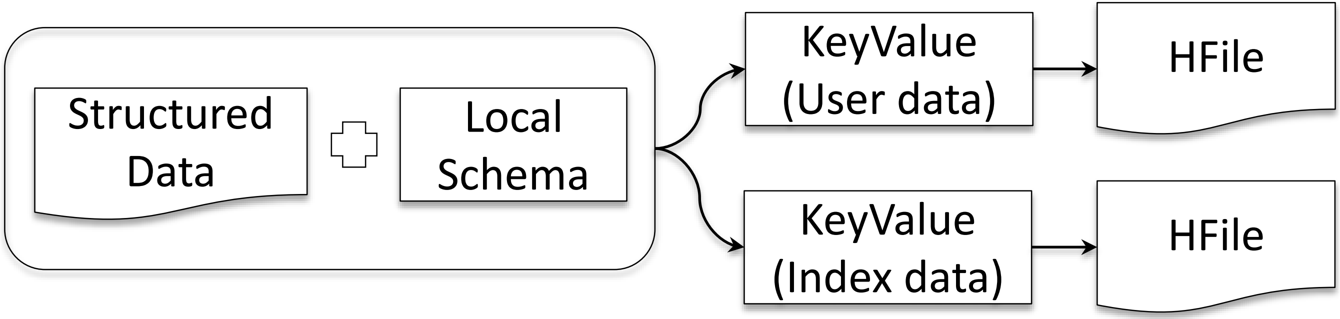
Result Scanner

Result

```
{
        table:   “TableA",
        conditions: [ “ID":  “23470%",  “CarNo":  “A1?234",
                              “Color” :  “Yello || Black || White”  ],
        columns: ["ID",  “Time",  “CarNo",  “Color”  ],
        caching: 100
}
```

- **Flexible and powerful query API**.
- Support for below operators:
  Range Query Operator:   >, >=, <, <=
  Logic Operator: &&, ||
  Fuzzy Query Operator: ?, *, %
- Index name can be specified, or just depend on imbedded RBO to choose the best index.
- Using exist or customized filters to push down queries for decreasing query latency.

# Bulk Load



- Schema has been defined in advance, including columns, column to qualifier mappings, index row key format, etc.  The only required configuration for bulk load task is the column orders of the data file.
- Secondary index related HFiles can be generated together in one bulk load task.

# Future Work For CTBase

1. Better Full-Text index support.
2. Active-Active Clusters Client.
3. Better HFile format for structured data.

# Agenda

CTBase: A  light weight HBase client for structured data

**Tagram: Distributed Bitmap index implementation with HBase**

CloudTable service(HBase on Huawei Cloud)

# Tagram Design Motivation

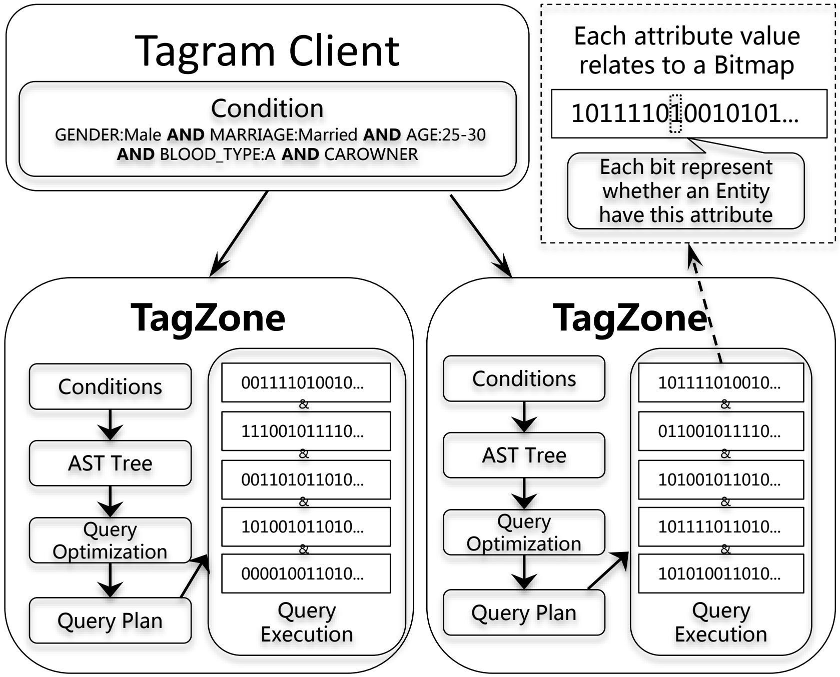
- Low-cardinality attributes are popularly used in Personas area, these attributes are used to describe user/entity typical characteristics, behavior patterns, motivations. E.g. Attributes for describing buyer personas can help identify where your best customers spend time on the internet.
- Ad-hoc queries must be supported. Likes:

  *“How many male customers have age < 30?”*

  *“How many customers have these specific attributes?”*

  *“Which people appeared in Area-A, Area-B and Area-C between 9:00 and 12:00?”*
- Solr/Elasticsearch based solutions are **not fast enough** for low-cardinality attributes based ad-hoc queries.

# Tagram Introduction



Tagram Client

Condition

GENDER:Male **AND** MARRIAGE:Married **AND** AGE:25-30
**AND** BLOOD_TYPE:A **AND** CAROWNER

Each attribute value relates to a Bitmap

101111010010101...

Each bit represent whether an Entity have this attribute

TagZone

Conditions

AST Tree

Query Optimization

Query Plan

001111010010...
&
111001011110...
&
001101011010...
&
101001011010...
&
000010011010...

Query Execution

TagZone

Conditions

AST Tree

Query Optimization

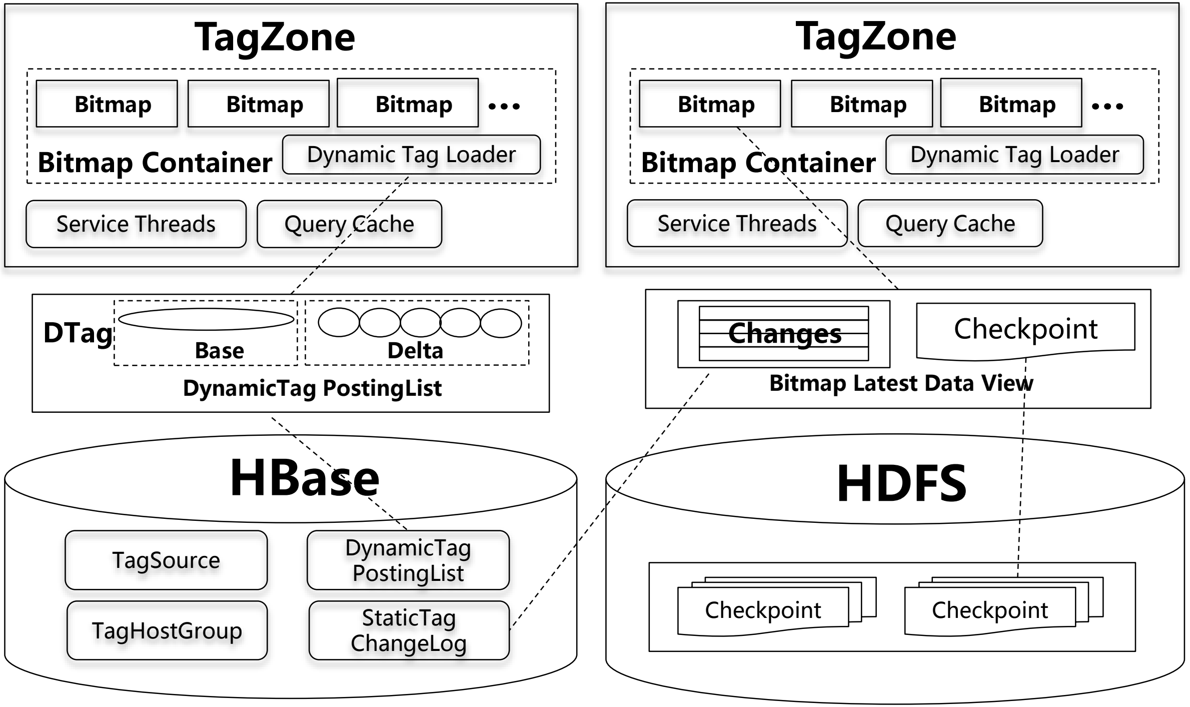
Query Plan

101111010010...
&
011001011110...
&
101001011010...
&
101111011010...
&
101010011010...

Query Execution

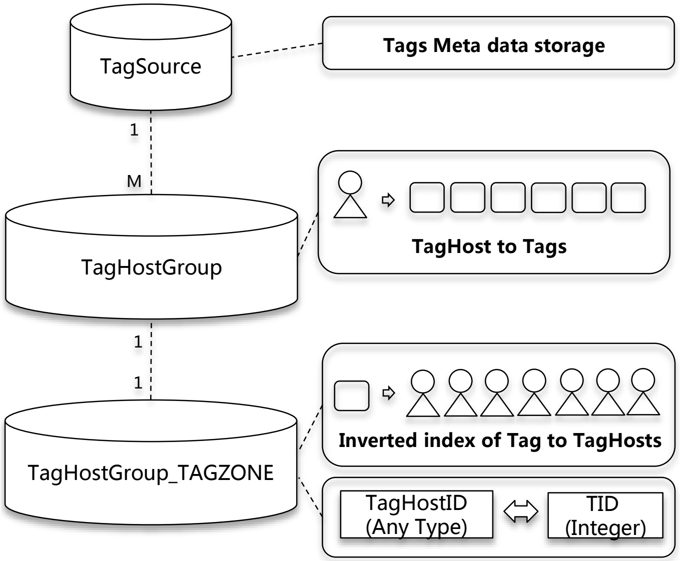
- **Distributed bitmap index** implementation uses HBase as backend storage.
- **Milliseconds level latency** for attribute based ad-hoc queries.
- Each attribute value is called a Tag. Entity is called a TagHost. Each Tag relates to an independent bitmap. Hot tags related bitmaps are memory-resident.
- A Tag is either static or dynamic. Static tags must be defined in advance. Dynamic tags have no such restriction, likes Time-Space related tags.

# Tagram Architecture



- TagZone service is initialized by HBase coprocessor.
- Each TagZone is an independent bitmap computing unit.
- All the real-time writes and logs are stored in HBase.
- Use bitmap checkpoint for fast recovery during service initialization.

# Data Model



- TagSource: Meta data storage for static tags, includes configurations per tag.
- TagHostGroup: Uses TagHostID as key, and store all the tags as columns.
- TagZone:  Inverted index from Tag to TagHost list. Bitmap related data is also stored in this table. Partitions are decided during table creation, and can not split in future.
- Each table is an independent HBase table.

# Query

**Query grammar in BNF**:

```
Query ::= ( Clause )+
Clause ::= ["AND", "OR", "NOT"] ([TagName:]TagValue| "(" Query ")" )
```

- A Query is a series of Clauses. Each Clause can also be a nested query.
- Supports AND/OR/NOT operators. AND indicates this clause is required, NOT indicates this clause is prohibited, OR indicates this clause should appear in the matching results. The default operator is OR is none operator specified.
- Parentheses “(” “)” can be used to improve the priority of a sub-query.

# Query Example

- Normal Query:

  GENDER:Male **AND** MARRIAGE:Married **AND** AGE:25-30 **AND** BLOOD_TYPE:A

- Use parentheses "(" ")" to improve the priority of sub-query:

  GENDER:Male **AND** MARRIAGE:Married **AND** (AGE:25-30 **OR** AGE:30-35) **AND** BLOOD_TYPE:A

- Minimum Number Should Match Query:

  At least 2 of below 4 groups of conditions should be satisfied:

  (A1 B1 C1 D1 E1 F1 G1 H1) (A2 B2 C2 D2 E2 F2 G2 H2) (A3 B3 C3 D3 E3 F3 G3 H3) (A4 B4 C4 D4 E4 F4 G4 H4)

- Complex query with static and dynamic tags:

  GENDER:Male **AND** MARRIAGE:Married **AND** AGE:25-30 **AND** CAROWNER **AND** $D:DTag1 **AND** $D:DTag2

# Evaluation

## Bitmap in-memory and on-disk size :

| Bitmap Cardinality | In-memory Bytes | On-Disk Size Bytes |
|---|---|---|
| 5,000,000 | 15426632 | 10387402 |
| 10,000,000 | 29042504 | 20370176 |
| 50,000,000 | 140155632 | 99812920 |
| 100,000,000 | 226915200 | 198083304 |

NOTE: 1. Bitmap cardinality is the number of bit 1 from the bitmap in binary form.
2. The positions with bit 1 are random integers between 0 and Integer.Max.
3. The distribution of bit 1(In Bitmap binary form) and the range may affect the bitmap size.

## Test results on small cluster :

3 Huawei 2288 Servers(256GB Memory, Intel(R) Xeon(R) CPU E5-2618L v3 @2.30GHZ*2 SATA,4TB*14)

1.5 Billion TagHosts, ~60 static Tags per TagHost.

Query with 10 random tags(Hundreds of thousands satisfied results), count and only return first screen results. Average query latency: 60ms。

# Future Work For Tagram

1. Multiple TagZone Replica.
2. Async Tagram/HBase Client.
3. Better Bitmap Memory Management.
4. Integration with Graph/Full-Text index.

# Agenda

CTBase: A light weight HBase client for structured data

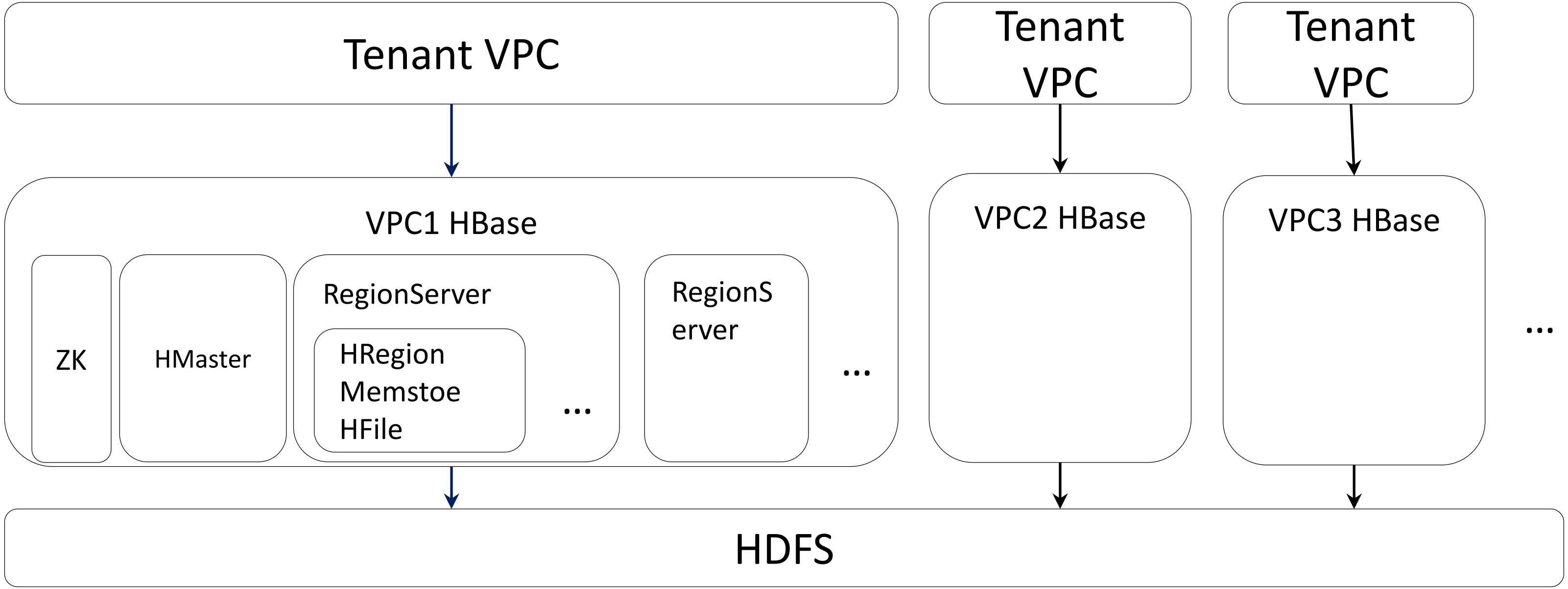
Tagram: Distributed Bitmap index implementation with HBase

**CloudTable service(HBase on Huawei Cloud)**

# CloudTable Service Features

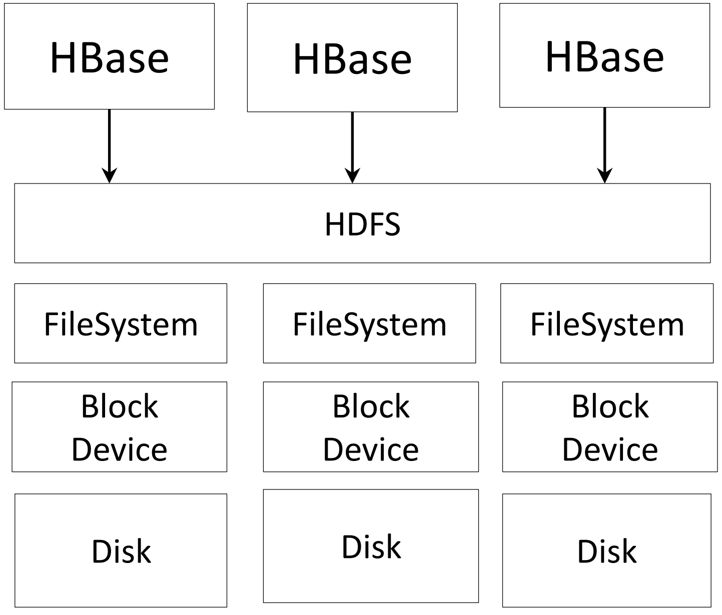
- Easy Maintenance
- Security
- High Performance
- SLA
- High Availability
- Low Cost

# CloudTable Service On Huawei Cloud

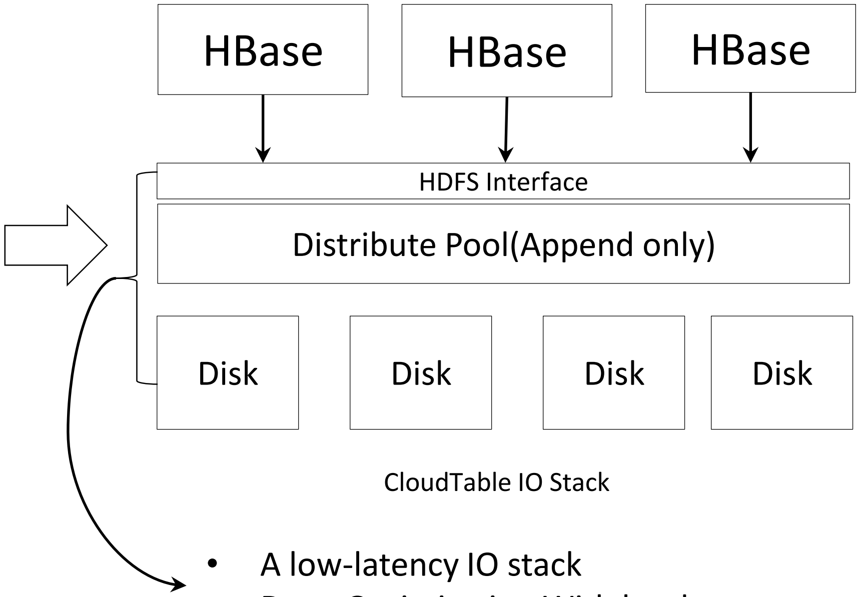


- Isolation by VPC
- Shared Storage

# CloudTable – IO Optimization



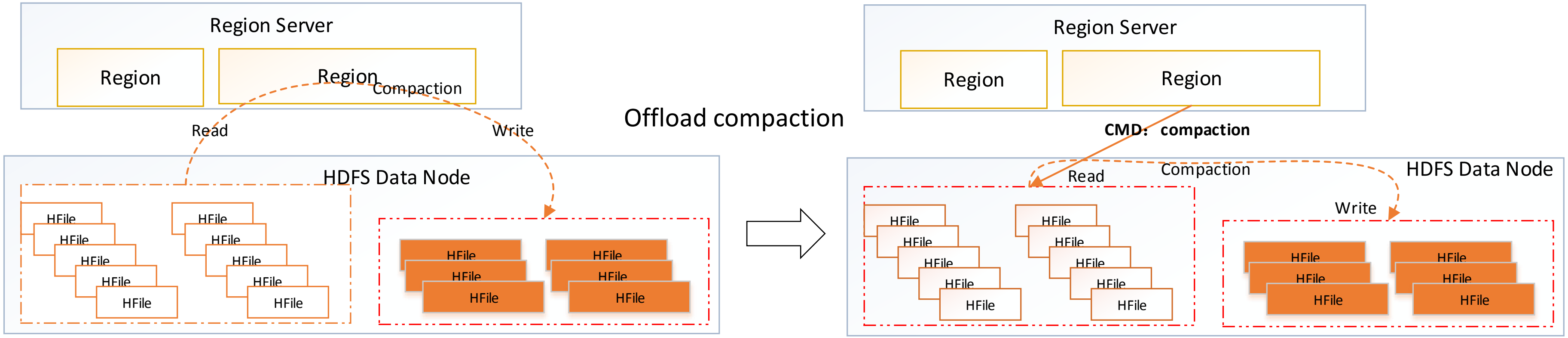
Native HBase IO Stack

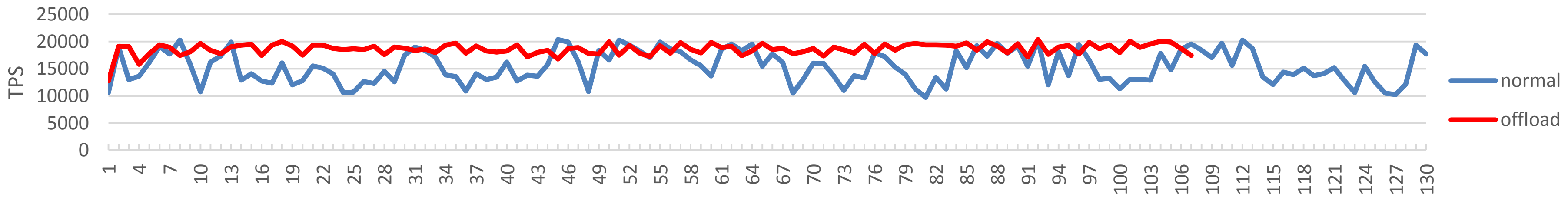


CloudTable IO Stack

- A low-latency IO stack
- Deep Optimization With hardware

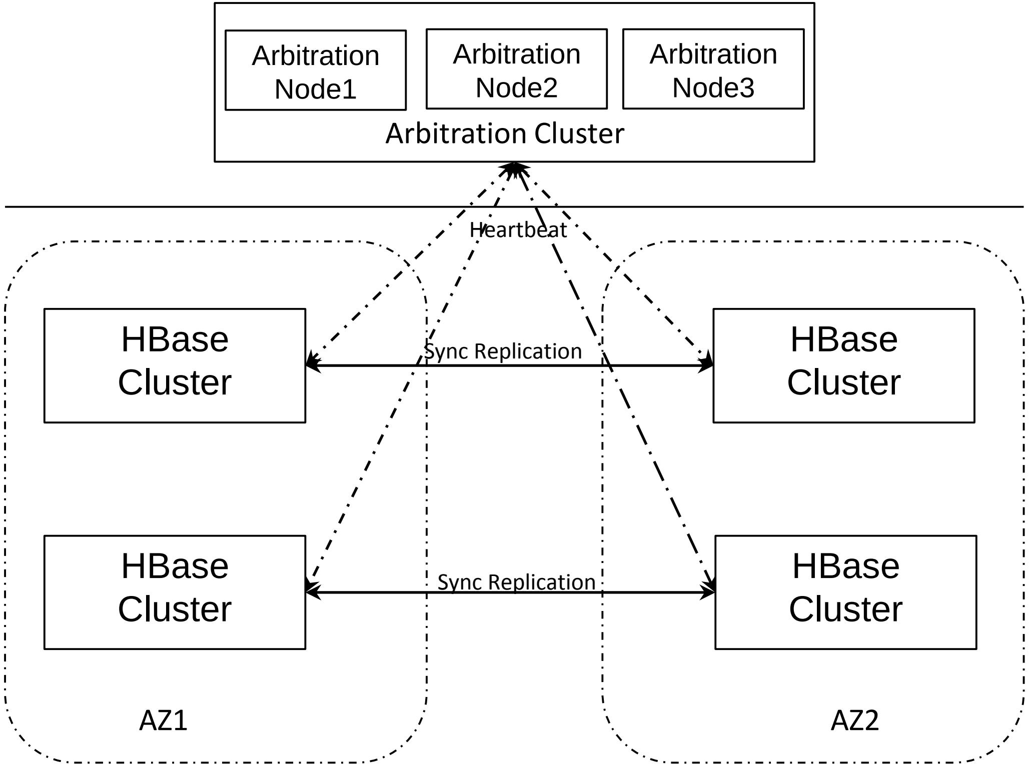
# CloudTable – Offload Compaction





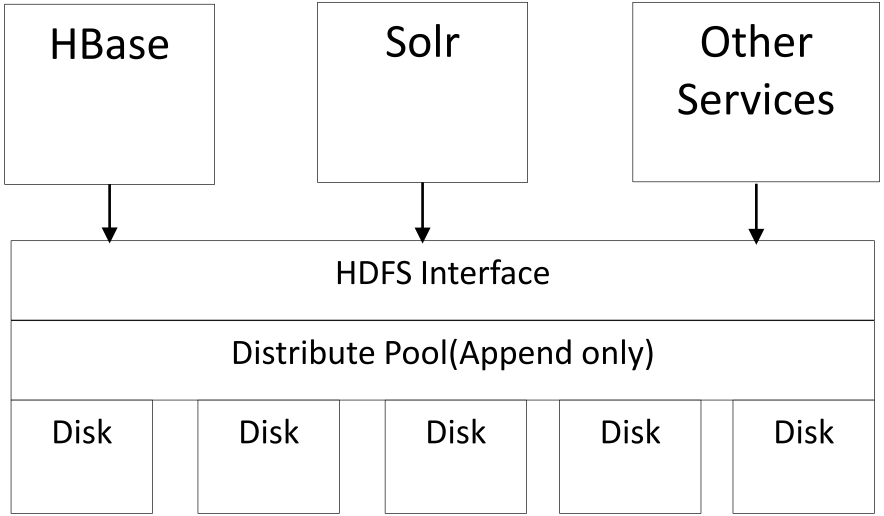
Smooth Performance

# CloudTable – High Availability

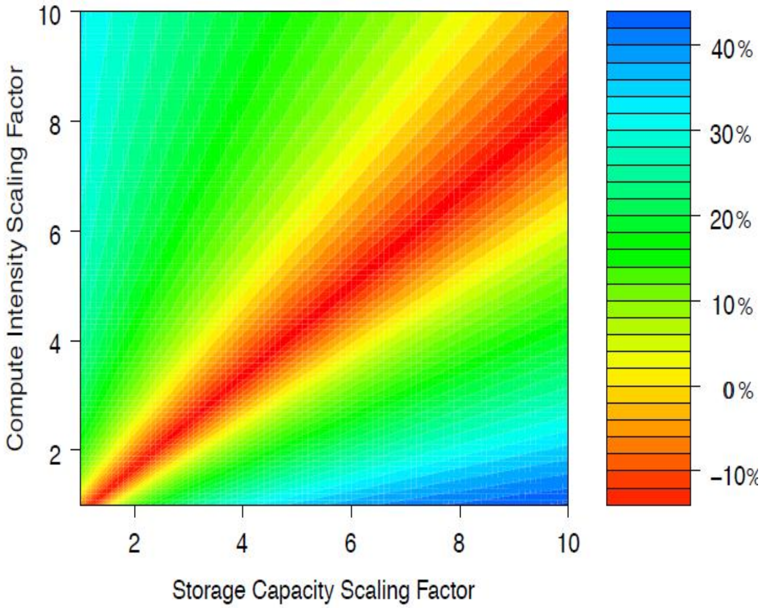


- Cross AZ Replication
- Write: Strong Consistency
- Read: Timeline Consistency
- 99.99% Availability
- 99.9999999 Durability
- Auto Failover

# CloudTable – Low Cost



- 40% resource savings



From:Flash Storage Disaggregation

# Thank You !

bijieshan@huawei.com

zhongyanhui@huawei.com