Architecture of Enterprise Applications 29
HBase

Haopeng Chen

RE*liable,* ***IN****telligent and* ***S****calable Systems Group (****REINS****)*

Shanghai Jiao Tong University

Shanghai, China

http://reins.se.sjtu.edu.cn/~chenhp

e-mail: chen-hp@sjtu.edu.cn

Contents and Objectives

- HBase
  - Basic Concepts
  - HBase Scheme Design
  - Other Issues

- Objectives
  - 能够根据大尺寸非结构化和半结构化数据存储与分析的要求，设计并实现基于HBase的数据存储与分析方案

Apache HBase

- Apache HBase™ is the Hadoop database, a distributed, scalable, big data store.
  - Use Apache HBase™ when you need random, realtime read/write access to your Big Data.
  - This project's goal is the hosting of very large tables -- billions of rows X millions of columns -- atop clusters of commodity hardware.
  - Apache HBase is an open-source, distributed, versioned, non-relational database modeled after Google's Bigtable: A Distributed Storage System for Structured Data by Chang et al.
  - Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

- https://hbase.apache.org/

Apache HBase

- HBase is a distributed column-oriented database built on top of HDFS.
  - HBase is the Hadoop application to use when you require real-time read/write random-access to very large datasets.
  - HBase comes at the scaling problem from the opposite direction.
    - It is built from the ground-up to scale linearly just by adding nodes.
  - HBase is not relational and does not support SQL, but given the proper problem space,
    - it is able to do what an RDBMS cannot: host very large, sparsely populated tables on clusters made from commodity hardware.
  - The canonical HBase use case is the webtable, a table of crawled web pages and their attributes (such as language and MIME type) keyed by the web page URL.
    - The webtable is large, with row counts that run into the billions.

Apache HBase

- **Features**
  - Linear and modular scalability.
  - Strictly consistent reads and writes.
  - Automatic and configurable sharding of tables
  - Automatic failover support between RegionServers.
  - Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables.
  - Easy to use Java API for client access.
  - Block cache and Bloom Filters for real-time queries.
  - Query predicate push down via server side Filters
  - Thrift gateway and a REST-ful Web service that supports XML, Protobuf, and binary data encoding options
  - Extensible jruby-based (JIRB) shell
  - Support for exporting metrics via the Hadoop metrics subsystem to files or Ganglia; or via JMX

Apache HBase

- Applications store data into labeled tables.
  - Tables are made of rows and columns.
  - Table cells—the intersection of row and column coordinates—are versioned.
    - By default, their version is a timestamp auto-assigned by HBase at the time of cell insertion.
    - A cell's content is an uninterpreted array of bytes.
  - Table row keys are also byte arrays,
    - so theoretically anything can serve as a row key from strings to binary representations of long or even serialized data structures.
  - Table rows are sorted by row key, the table's primary key.
    - The sort is byte-ordered.
    - All table accesses are via the table primary key.

Apache HBase

- Applications store data into labeled tables.
  - Row columns are grouped into column families.
    - All column family members have a common prefix, so, for example, the columns temperature:air and temperature:dew_point are both members of the temperature column family, whereas station:identifier belongs to the station family.
  - The column family prefix must be composed of printable characters.
    - The qualifying tail, the column family qualifier, can be made of any arbitrary bytes.
  - A table's column families must be specified up front as part of the table schema definition,
    - but new column family members can be added on demand.
  - In synopsis, HBase tables are like those in an RDBMS,
    - only cells are versioned, rows are sorted, and columns can be added on the fly by the client as long as the column family they belong to preexists.

Apache HBase

- Regions
  - Tables are automatically partitioned horizontally by HBase into regions.
    - Each region comprises a subset of a table's rows.
    - A region is denoted by the table it belongs to, its first row, inclusive, and last row, exclusive.
    - Initially, a table comprises a single region, but as the size of the region grows, after it crosses a configurable size threshold, it splits at a row boundary into two new regions of approximately equal size.
    - Until this first split happens, all loading will be against the single server hosting the original region.
    - As the table grows, the number of its regions grows. Regions are the units that get distributed over an HBase cluster.
- Locking
  - Row updates are atomic, no matter how many row columns constitute the row-level transaction.
    - This keeps the locking model simple.

Get Start with HBase

REliable, INtelligent & Scalable Systems

- Set the JAVA_HOME environment in *conf/hbase-env.sh* file.

```
# Set environment variables here.
# The java implementation to use.
export JAVA_HOME=$(/usr/libexec/java_home)
```

- Edit *conf/hbase-site.xml*

```
<configuration>
  <property>
    <name>hbase.rootdir</name>
    <value>file:///Users/user/hbase</value>
  </property>
</configuration>
```

Get Start with HBase

REliable, INtelligent & Scalable Systems

- The *bin/start-hbase.sh* script is provided as a convenient way to start HBase.
- You can use the jps command to verify that you have one running process called HMaster.

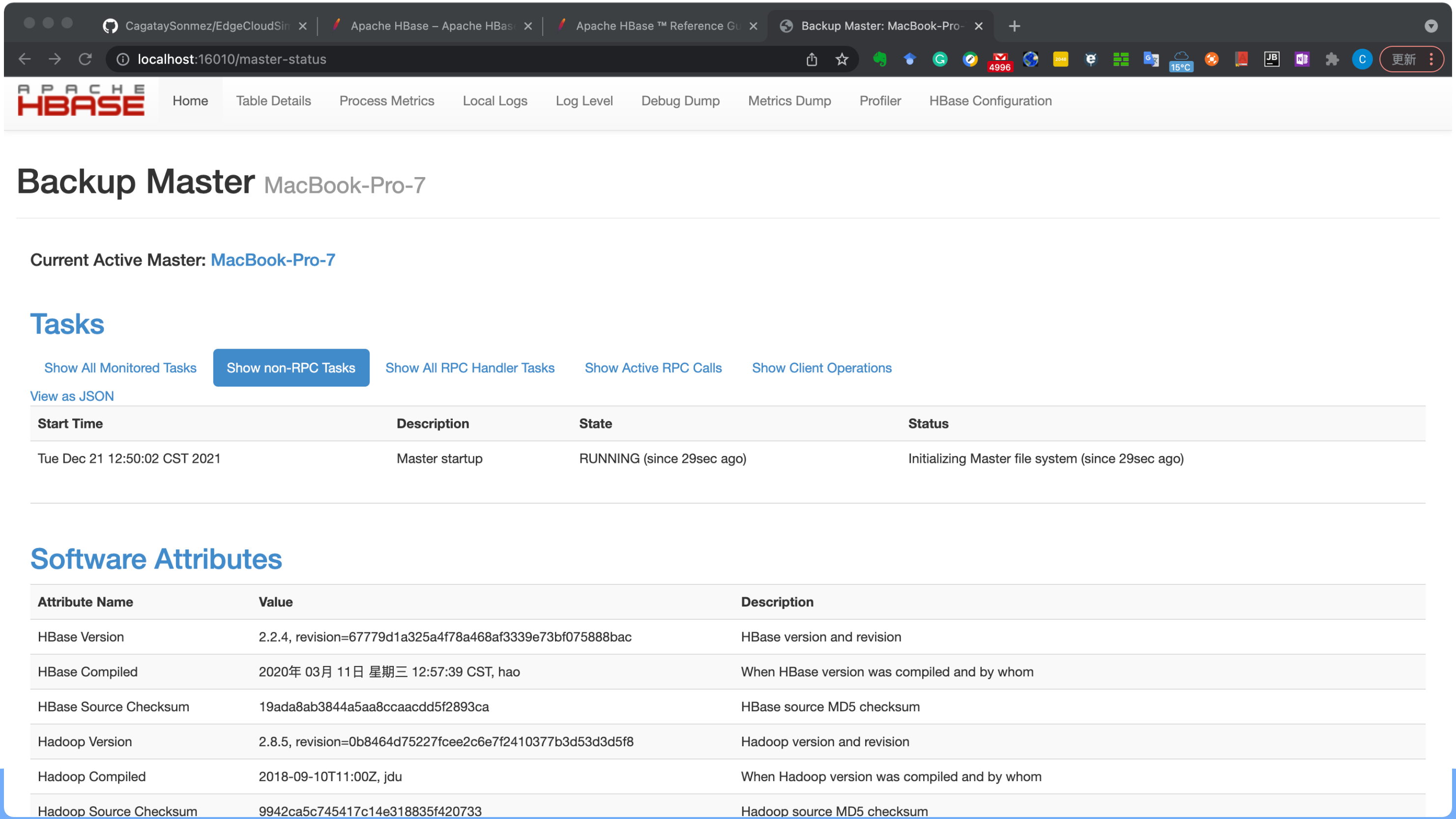
```
hbase-2.4.8 % jps
71047 HMaster
59738
71149 Jps
59199
```

- In standalone mode HBase runs all daemons within this single JVM, i.e.
  - the HMaster, a single HRegionServer, and the ZooKeeper daemon.

Get Start with HBase

REliable, INtelligent & Scalable Systems

- Go to http://localhost:16010 to view the HBase Web UI.



CagataySonmez/EdgeCloudSim
Apache HBase – Apache HBase
Apache HBase ™ Reference Gu
Backup Master: MacBook-Pro-
localhost:16010/master-status
更新
Home
Table Details
Process Metrics
Local Logs
Log Level
Debug Dump
Metrics Dump
Profiler
HBase Configuration

Backup Master MacBook-Pro-7

Current Active Master: MacBook-Pro-7

Tasks

Show All Monitored Tasks | Show non-RPC Tasks | Show All RPC Handler Tasks | Show Active RPC Calls | Show Client Operations

View as JSON

Start Time	Description	State	Status
Tue Dec 21 12:50:02 CST 2021	Master startup	RUNNING (since 29sec ago)	Initializing Master file system (since 29sec ago)

Software Attributes

Attribute Name	Value	Description
HBase Version	2.2.4, revision=67779d1a325a4f78a468af3339e73bf075888bac	HBase version and revision
HBase Compiled	2020年 03月 11日 星期三 12:57:39 CST, hao	When HBase version was compiled and by whom
HBase Source Checksum	19ada8ab3844a5aa8ccaacdd5f2893ca	HBase source MD5 checksum
Hadoop Version	2.8.5, revision=0b8464d75227fcee2c6e7f2410377b3d53d3d5f8	Hadoop version and revision
Hadoop Compiled	2018-09-10T11:00Z, jdu	When Hadoop version was compiled and by whom
Hadoop Source Checksum	9942ca5c745417c14e318835f420733	Hadoop source MD5 checksum

Get Start with HBase

REliable, INtelligent & Scalable Systems

- Connect to HBase.
  - Connect to your running instance of HBase using the hbase shell command, located in the *bin/* directory of your HBase install.
  - In this example, some usage and version information that is printed when you start HBase Shell has been omitted. The HBase Shell prompt ends with a > character.

```
$ hbase-2.4.8 % ./bin/hbase shell
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.2.4, r67779d1a325a4f78a468af3339e73bf075888bac, 2020年 03月 11日 星期
三 12:57:39 CST
Took 0.0019
seconds

hbase(main):001:0>
```

Get Start with HBase

- Create a table.
  - Use the create command to create a new table. You must specify the table name and the ColumnFamily name.

```
hbase(main):001:0> create 'test', 'cf'
0 row(s) in 0.4170 seconds
=> Hbase::Table - test
```

- List Information About your Table
  - Use the list command to confirm your table exists

```
hbase(main):002:0> list 'test'
TABLE
test
1 row(s) in 0.0180 seconds
=> ["test"]
```

Get Start with HBase

- Now use the describe command to see details, including configuration defaults

```
hbase(main):003:0> describe 'test'
Table test is ENABLED
test
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf', VERSIONS => '1', EVICT_BLOCKS_ON_CLOSE => 'false', NEW_VERSION_BEHAVIOR =>
'false', KEEP_DELETED_CELLS => 'FALSE', CACHE_DATA_ON_WRITE => 'false',
DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', MIN_VERSIONS => '0', REPLICATION_SCOPE
=> '0', BLOOMFILTER => 'ROW', CACHE_INDEX_ON_WRITE => 'f alse', IN_MEMORY => 'false',
CACHE_BLOOMS_ON_WRITE => 'false', PREFETCH_BLOCKS_ON_OPEN => 'false', COMPRESSION =>
'NONE', BLOCKCACHE => 'true', BLOCKSIZE => '65536'}
1 row(s)
Took 0.9998 seconds
```

Get Start with HBase

- Put data into your table.
  - To put data into your table, use the put command.

```
hbase(main):003:0> put 'test', 'row1', 'cf:a', 'value1'
0 row(s) in 0.0850 seconds
hbase(main):004:0> put 'test', 'row2', 'cf:b', 'value2'
0 row(s) in 0.0110 seconds
hbase(main):005:0> put 'test', 'row3', 'cf:c', 'value3'
0 row(s) in 0.0100 seconds
```

  - Here, we insert three values, one at a time.
  - The first insert is at row1, column cf:a, with a value of value1.
  - Columns in HBase are comprised of a column family prefix, cf in this example, followed by a colon and then a column qualifier suffix, a in this case.

Get Start with HBase

- Scan the table for all data at once.
  - One of the ways to get data from HBase is to scan.
  - Use the scan command to scan the table for data.
  - You can limit your scan, but for now, all data is fetched.

```
hbase(main):006:0> scan 'test'
ROW       COLUMN+CELL
  row1    column=cf:a, timestamp=1421762485768, value=value1
  row2    column=cf:b, timestamp=1421762491785, value=value2
  row3    column=cf:c, timestamp=1421762496210, value=value3
3 row(s) in 0.0230 seconds
```

Get Start with HBase

REliable, INtelligent & Scalable Systems

- Get a single row of data.
  - To get a single row of data at a time, use the get command.

```
hbase(main):007:0> get 'test', 'row1'
COLUMN                 CELL
 cf:a                  timestamp=1421762485768, value=value1
1 row(s) in 0.0350 seconds
```

Get Start with HBase

REliable, INtelligent & Scalable Systems

- Disable a table.
  - If you want to delete a table or change its settings, as well as in some other situations, you need to disable the table first, using the disable command.
  - You can re-enable it using the enable command.

```
hbase(main):008:0> disable 'test'
0 row(s) in 1.1820 seconds

hbase(main):009:0> enable 'test'
0 row(s) in 0.1770 seconds
```

  - Disable the table again if you tested the enable command above:

```
hbase(main):010:0> disable 'test'
0 row(s) in 1.1820 seconds
```

Get Start with HBase

- Drop the table.
  - To drop (delete) a table, use the drop command.

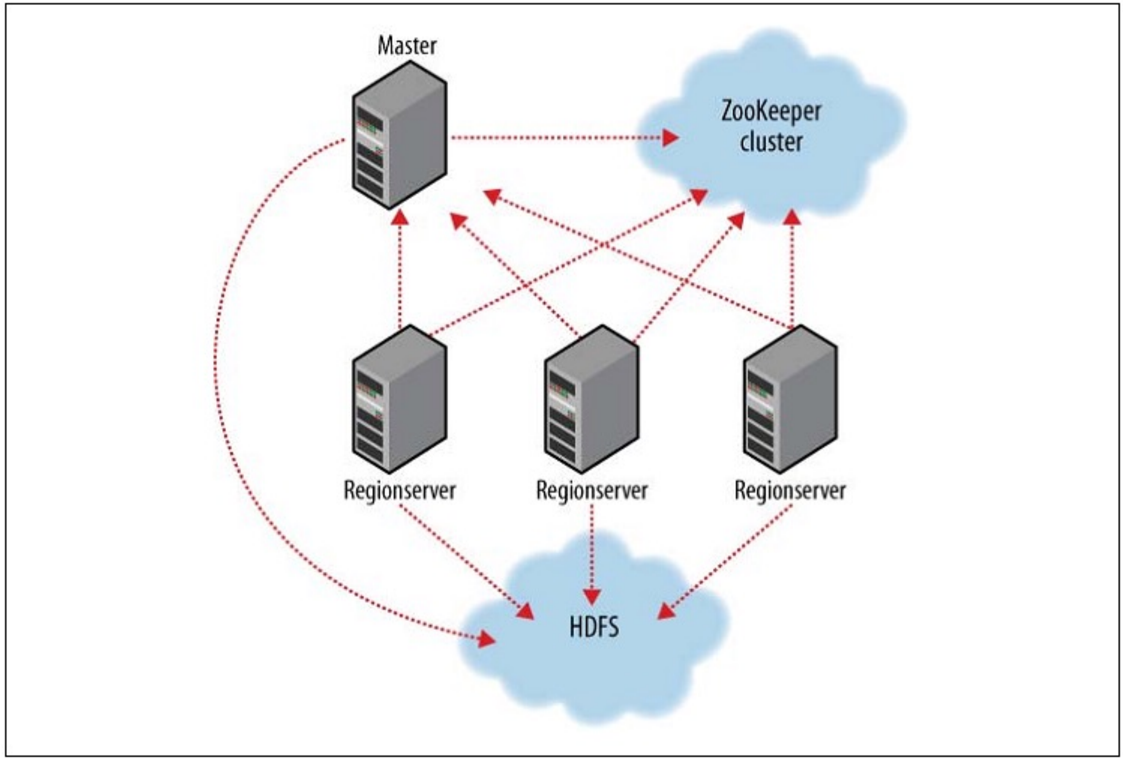
```
hbase(main):011:0> drop 'test'
0 row(s) in 0.1370 seconds
```

- Exit the HBase Shell.
  - To exit the HBase Shell and disconnect from your cluster, use the quit command. HBase is still running in the background.

- *Stop Hbase*

```
$ ./bin/stop-hbase.sh
stopping hbase....................
$
```

HBase cluster members



Data Model

- In HBase, data is stored in tables, which have rows and columns.
  - This is a terminology overlap with relational databases (RDBMSs), but this is not a helpful analogy. Instead, it can be helpful to think of an HBase table as a multi-dimensional map.

- *HBase Data Model Terminology*
  - **Table**: An HBase table consists of multiple rows.
  - **Row**: A row in HBase consists of a row key and one or more columns with values associated with them.
    - Rows are sorted alphabetically by the row key as they are stored.
    - For this reason, the design of the row key is very important. The goal is to store data in such a way that related rows are near each other.
    - A common row key pattern is a website domain. If your row keys are domains, you should probably store them in reverse (org.apache.www, org.apache.mail, org.apache.jira). This way, all of the Apache domains are near each other in the table, rather than being spread out based on the first letter of the subdomain.

Data Model

- In HBase, data is stored in tables, which have rows and columns.
  - This is a terminology overlap with relational databases (RDBMSs), but this is not a helpful analogy. Instead, it can be helpful to think of an HBase table as a multi-dimensional map.

- *HBase Data Model Terminology*
  - **Column**: A column in HBase consists of a column family and a column qualifier, which are delimited by a : (colon) character.
  - **Column Family**: Column families physically colocate a set of columns and their values, often for performance reasons.
    - Each column family has a set of storage properties, such as whether its values should be cached in memory, how its data is compressed or its row keys are encoded, and others.
    - Each row in a table has the same column families, though a given row might not store anything in a given column family.

Data Model

- In HBase, data is stored in tables, which have rows and columns.
  - This is a terminology overlap with relational databases (RDBMSs), but this is not a helpful analogy. Instead, it can be helpful to think of an HBase table as a multi-dimensional map.

- *HBase Data Model Terminology*
  - **Column Qualifier**: A column qualifier is added to a column family to provide the index for a given piece of data.
    - Given a column family content, a column qualifier might be content:html, and another might be content:pdf. Though column families are fixed at table creation, column qualifiers are mutable and may differ greatly between rows.
  - **Cell**: A cell is a combination of row, column family, and column qualifier, and contains a value and a timestamp, which represents the value’s version.
  - **Timestamp**: A timestamp is written alongside each value, and is the identifier for a given version of a value. By default, the timestamp represents the time on the RegionServer when the data was written, but you can specify a different timestamp value when you put data into the cell.

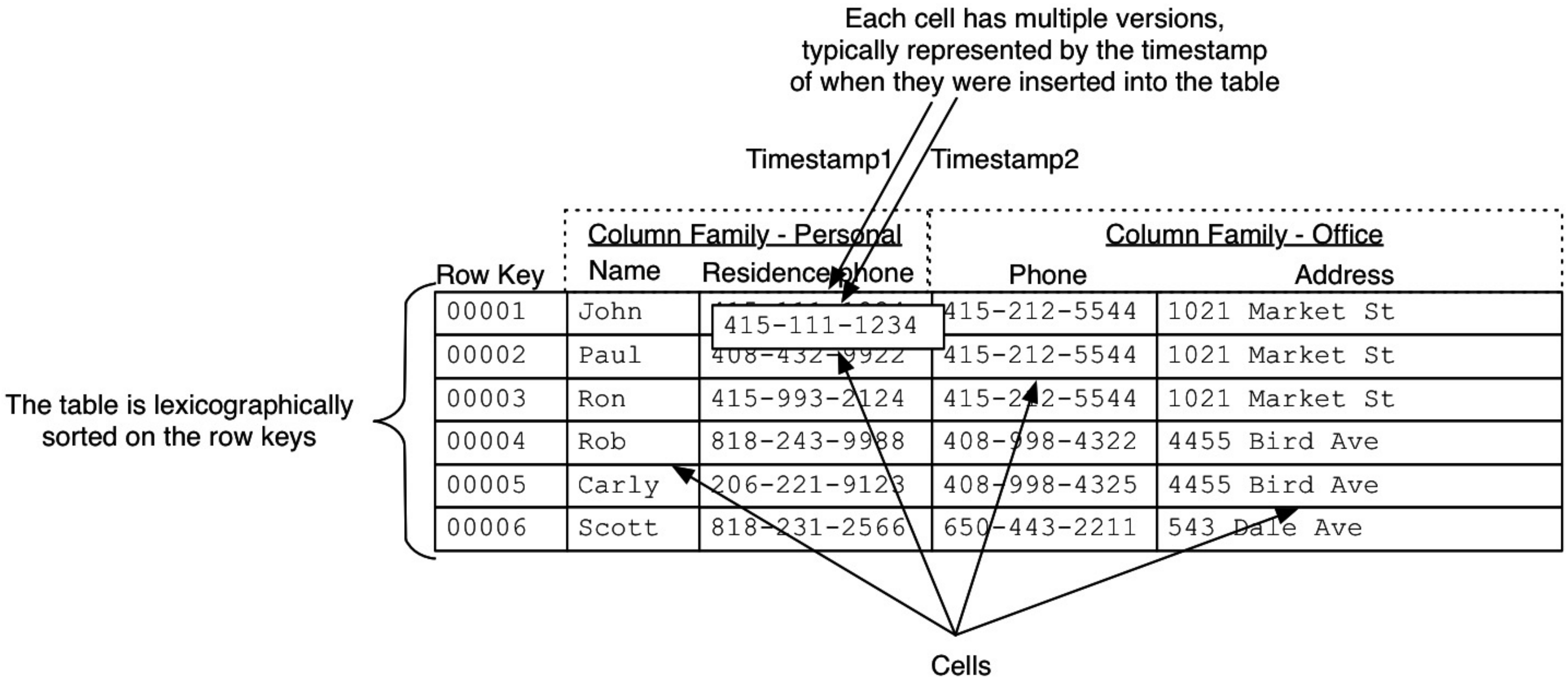
Conceptual View

- Cells in this table that appear to be empty do not take space, or in fact exist, in HBase.
- This is what makes HBase "sparse."

Row Key	Time Stamp	ColumnFamily contents	ColumnFamily anchor	ColumnFamily people
"com.cnn.www"	t9		anchor:cnnsi.com = "CNN"	
"com.cnn.www"	t8		anchor:my.look.ca = "CNN.com"	
"com.cnn.www"	t6	contents:html = "<html>..."		
"com.cnn.www"	t5	contents:html = "<html>..."		
"com.cnn.www"	t3	contents:html = "<html>..."		
"com.example.www"	t5	contents:html = "<html>..."		people:author = "John Doe"

Conceptual View

- Cells in this table that appear to be empty do not take space, or in fact exist, in HBase.
- This is what makes HBase "sparse."

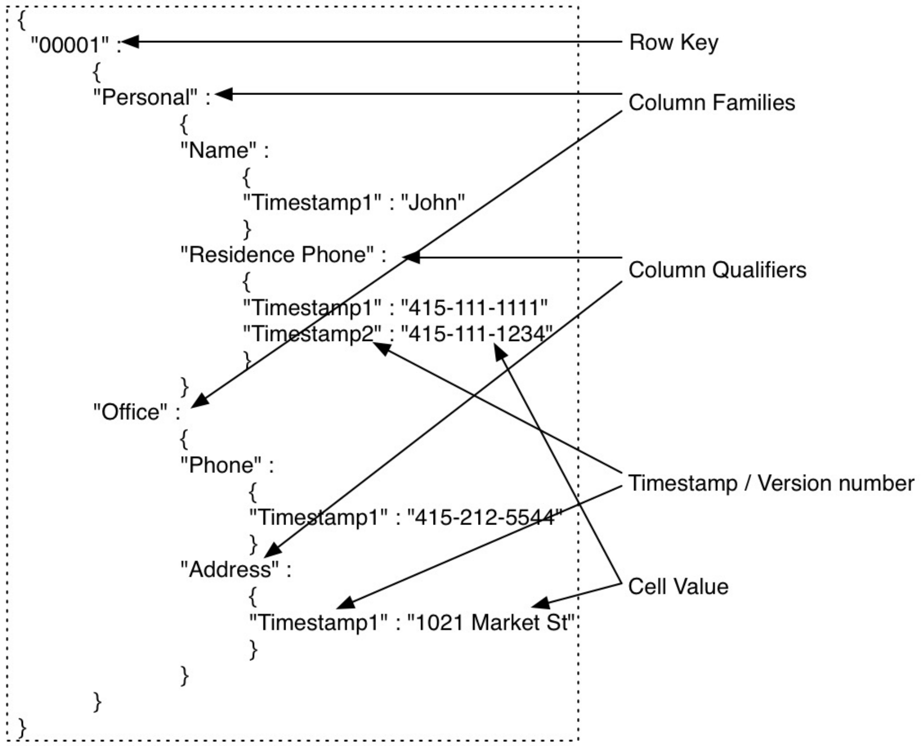

Each cell has multiple versions, typically represented by the timestamp of when they were inserted into the table
Timestamp1
Timestamp2
Column Family - Personal
Column Family - Office
The table is lexicographically sorted on the row keys
Cells

Row Key	Name	Residence phone	Phone	Address
00001	John	415-111-1234	415-212-5544	1021 Market St
00002	Paul	408-432-9922	415-212-5544	1021 Market St
00003	Ron	415-993-2124	415-212-5544	1021 Market St
00004	Rob	818-243-9988	408-998-4322	4455 Bird Ave
00005	Carly	206-221-9123	408-998-4325	4455 Bird Ave
00006	Scott	818-231-2566	650-443-2211	543 Dale Ave

Conceptual View

- The following represents the same information as a multi-dimensional map.

```
{
 "com.cnn.www": {
   contents: {
    t6: contents:html: "<html>..."
    t5: contents:html: "<html>..."
    t3: contents:html: "<html>..."
   }
   anchor: {
    t9: anchor:cnnsi.com = "CNN"
    t8: anchor:my.look.ca = "CNN.com"
   }
   people: {}
 }
 "com.example.www": {
   contents: {
    t5: contents:html: "<html>..."
   }
   anchor: {}
   people: {
    t5: people:author: "John Doe"
   }
  }
}
```



```
{
 "00001" :
    {
    "Personal" :
         {
         "Name" :
              {
              "Timestamp1" : "John"
              }
         "Residence Phone" :
              {
              "Timestamp1" : "415-111-1111"
              "Timestamp2" : "415-111-1234"
              }
         }
    "Office" :
         {
         "Phone" :
              {
              "Timestamp1" : "415-212-5544"
              }
         "Address" :
              {
              "Timestamp1" : "1021 Market St"
              }
         }
    }
}
```

Physical View

- *ColumnFamily* anchor

Row Key	Time Stamp	Column Family anchor
"com.cnn.www"	t9	anchor:cnnsi.com = "CNN"
"com.cnn.www"	t8	anchor:my.look.ca = "CNN.com"

- *ColumnFamily* contents

Row Key	Time Stamp	Column Family contents
"com.cnn.www"	t6	contents:html = "<html>..."
"com.cnn.www"	t5	contents:html = "<html>..."
"com.cnn.www"	t3	contents:html = "<html>..."

Namespace

- A namespace is a logical grouping of tables analogous to a database in relation database systems. This abstraction lays the groundwork for upcoming multi-tenancy related features:
  - Quota Management (HBASE-8410) - Restrict the amount of resources (i.e. regions, tables) a namespace can consume.
  - Namespace Security Administration (HBASE-9206) - Provide another level of security administration for tenants.
  - Region server groups (HBASE-6721) - A namespace/table can be pinned onto a subset of RegionServers thus guaranteeing a coarse level of isolation.

Namespace

- A namespace is a logical grouping of tables analogous to a database in relation database systems. This abstraction lays the groundwork for upcoming multi-tenancy related features:
  - Quota Management (HBASE-8410) - Restrict the amount of resources (i.e. regions, tables) a namespace can consume.
  - Namespace Security Administration (HBASE-9206) - Provide another level of security administration for tenants.
  - Region server groups (HBASE-6721) - A namespace/table can be pinned onto a subset of RegionServers thus guaranteeing a coarse level of isolation.
- A namespace can be created, removed or altered.
  - Namespace membership is determined during table creation by specifying a fully-qualified table name of the form:
  - `<table namespace>:<table qualifier>`

Namespace

```
#Create a namespace
create_namespace 'my_ns'

#create my_table in my_ns namespace
create 'my_ns:my_table', 'fam’

#drop namespace
drop_namespace 'my_ns’

#alter namespace
alter_namespace 'my_ns', {METHOD => 'set', 'PROPERTY_NAME' =>
'PROPERTY_VALUE'}
```

Namespace

- Predefined namespaces
  - There are two predefined special namespaces:
  - hbase - system namespace, used to contain HBase internal tables
  - default - tables with no explicit specified namespace will automatically fall into this namespace

```
#namespace=foo and table qualifier=bar
create 'foo:bar', 'fam’

#namespace=default and table qualifier=bar
create 'bar', 'fam'
```

Data Model Operations

- Get
  - Get returns attributes for a specified row. Gets are executed via Table.get
- Put
  - Put either adds new rows to a table (if the key is new) or can update existing rows (if the key already exists). Puts are executed via Table.put (non-writeBuffer) or Table.batch (non-writeBuffer)
- Scans
  - Scan allow iteration over multiple rows for specified attributes.
- Delete
  - Delete removes a row from a table. Deletes are executed via Table.delete.

HBase Clients

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;


public class HBaseTest {
    //获取配置信息
    public static Configuration conf;

    static{
        conf = HBaseConfiguration.create();
    }
```

HBase Clients

```
//1.判断一张表是否存在
public static boolean isExist(String tableName){
    //对表操作需要使用HbaseAdmin
    try {
        Connection connection = ConnectionFactory.createConnection(conf);
        //管理表
        HBaseAdmin admin = (HBaseAdmin) connection.getAdmin();

        return admin.tableExists(TableName.valueOf(tableName));
    } catch (IOException e) {
        e.printStackTrace();
    }
    return false;
}
```

HBase Clients

REliable, INtelligent & Scalable Systems

```
//2.在hbase创建表
public static void createTable(String tableName,String... columnfamily){
    try {
        //对表操作需要使用HbaseAdmin
        Connection connection = ConnectionFactory.createConnection(conf);
        //管理表
        HBaseAdmin admin = (HBaseAdmin) connection.getAdmin();
        //1.表如果存在，请输入其他表名
        if(isExist(tableName)){
            System.out.println("表存在，请输入其他表名");
        }else{
            //2.注意:创建表的话，需要创建一个描述器
            HTableDescriptor htd = new HTableDescriptor(TableName.valueOf(tableName));
            //3.创建列族
            for(String cf:columnfamily){
                htd.addFamily(new HColumnDescriptor(cf));
            }
            //4.创建表
            admin.createTable(htd);
            System.out.println("表已经创建成功！");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

HBase Clients

```
//3.删除hbase的表
public static void deleteTable(String tableName) {
    try {
        //对表操作需要使用HbaseAdmin
        Connection connection = ConnectionFactory.createConnection(conf);
        //管理表
        HBaseAdmin admin = (HBaseAdmin) connection.getAdmin();

        //1.表如果存在，请输入其他表名
        if (!isExist(tableName)) {
            System.out.println("表不存在");
        } else {
            //2.如果表存在，删除
            admin.disableTable(TableName.valueOf(tableName));
            admin.deleteTable(TableName.valueOf(tableName));
            System.out.println("表删除了");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

HBase Clients

```
//4.添加数据put 'user','rowkey','info:name','tony'
public static void addRow(String tableName,String rowkey,String cf,String column,String value){
    try {
        //对表操作需要使用HbaseAdmin
        Connection connection = ConnectionFactory.createConnection(conf);
        Table t = connection.getTable(TableName.valueOf(tableName));
        //1.表如果存在，请输入其他表名
        if (!isExist(tableName)) {
            System.out.println("表不存在");
        } else {
            //2.用put方式加入数据
            Put p = new Put(Bytes.toBytes(rowkey));
            //3.加入数据
            p.addColumn(Bytes.toBytes(cf),Bytes.toBytes(column),Bytes.toBytes(value));
            t.put(p);

        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

HBase Clients

REliable, INtelligent & Scalable Systems

```
//5.删除表中一行数据
public static void deleteRow(String tableName,String rowkey,String cf ){
    try {
        //对表操作需要使用HbaseAdmin
        Connection connection = ConnectionFactory.createConnection(conf);
        Table t = connection.getTable(TableName.valueOf(tableName));
        //1.表如果存在，请输入其他表名
        if (!isExist(tableName)) {
            System.out.println("表不存在");
        } else {
            //1.根据rowkey删除数据
            Delete delete = new Delete(Bytes.toBytes(rowkey));
            //2.删除
            t.delete(delete);
            System.out.println("删除成功");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

HBase Clients

REliable, INtelligent & Scalable Systems

```
//6.删除多行数据
public static void deleteAll(String tableName,String... rowkeys){
    try {
        //对表操作需要使用HbaseAdmin
        Connection connection = ConnectionFactory.createConnection(conf);
        Table t = connection.getTable(TableName.valueOf(tableName));
        //1.表如果存在，请输入其他表名
        if (!isExist(tableName)) {
            System.out.println("表不存在");
        } else {
            //1.把delete封装到集合
            List<Delete> list = new ArrayList<Delete>();
            //2.遍历
            for (String row:rowkeys){
                Delete d = new Delete(Bytes.toBytes(row));
                list.add(d);
            }
            t.delete(list);
            System.out.println("删除成功");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

HBase Clients

REliable, INtelligent & Scalable Systems

```
//7.扫描表数据 scan全表扫描
public static void scanAll(String tableName){
    try {
        //对表操作需要使用HbaseAdmin
        Connection connection = ConnectionFactory.createConnection(conf);
        Table t = connection.getTable(TableName.valueOf(tableName));

        //1.实例scan
        Scan s = new Scan();
        //2.拿到Scanner对象
        ResultScanner rs = t.getScanner(s);

        //3.遍历
        for (Result r:rs){
            Cell[] cells = r.rawCells();
            //遍历具体数据
            for (Cell c : cells){
                System.out.print("行键为: "+Bytes.toString(CellUtil.cloneRow(c))+"  ");
                System.out.print("列族为: "+Bytes.toString(CellUtil.cloneFamily(c))+"  ");
                System.out.print("列名为: "+Bytes.toString(CellUtil.cloneQualifier(c))+"  ");
                System.out.println("值为: "+Bytes.toString(CellUtil.cloneValue(c)));
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

HBase Clients

```
//8.获取一行表数据
public static void getRow(String tableName,String rowkey) throws IOException {
    Connection connection = ConnectionFactory.createConnection(conf);
    //拿到表对象
    Table t = connection.getTable(TableName.valueOf(tableName));

    //1.扫描指定数据需要实例对象Get
    Get get = new Get(Bytes.toBytes(rowkey));
    //2.可加过滤条件
    get.addFamily(Bytes.toBytes("info"));
    Result rs = t.get(get);
    //3.遍历
    Cell[] cells = rs.rawCells();
    for (Cell c : cells){
        System.out.print("行键为："+Bytes.toString(CellUtil.cloneRow(c))+"  ");
        System.out.print("列族为："+Bytes.toString(CellUtil.cloneFamily(c))+"  ");
        System.out.print("列名："+Bytes.toString(CellUtil.cloneQualifier(c))+"  ");
        System.out.println("值为："+Bytes.toString(CellUtil.cloneRow(c))+"  ");
    }
}
```

HBase Clients

```
    public static void main(String[] args) throws IOException {
        System.out.println(isExist("test"));
        createTable("test","info");
        deleteTable("test");
        addRow("test","101","info","age","20");
        deleteRow("test","101","info");
        deleteAll("test","1001","1002");
        scanAll("test");
        getRow("test","101");
    }
}
```

HBase Clients

- pom.xml

```
?xml version="1.0" encoding="UTF-8"?>
<project >
   <modelVersion>4.0.0</modelVersion>

   <groupId>org.example</groupId>
   <artifactId>SE3353_29_HBaseSample</artifactId>
   <version>1.0-SNAPSHOT</version>

   <dependencies>
      <!-- https://mvnrepository.com/artifact/org.apache.hbase/hbase-client -->
      <dependency>
         <groupId>org.apache.hbase</groupId>
         <artifactId>hbase-client</artifactId>
         <version>2.4.8</version>
      </dependency>
   </dependencies>
</project>
```

Versions

- A *{row, column, version}* tuple exactly specifies a cell in HBase.
  - It's possible to have an unbounded number of cells where the row and column are the same but the cell address differs only in its version dimension.
  - The HBase version dimension is stored in decreasing order, so that when reading from a store file, the most recent values are found first.
- Specifying the Number of Versions to Store
  - The maximum number of versions to store for a given column is part of the column schema and is specified at table creation, or via an alter command, via HColumnDescriptor.DEFAULT_VERSIONS.

  - *Modify the Maximum Number of Versions for a Column Family*
  - `hbase> alter ‘t1', NAME => ‘f1', VERSIONS => 5`
  - *Modify the Mimimum Number of Versions for a Column Family*
  - `hbase> alter ‘t1', NAME => ‘f1', MIN_VERSIONS => 5`

Versions

- By default, i.e. if you specify no explicit version, when doing a get, the cell whose version has the largest value is returned (which may or may not be the latest one written, see later).
- The default behavior can be modified in the following ways:
  - to return more than one version, see Get.setMaxVersions()
  - to return versions other than the latest, see Get.setTimeRange()

```
public static final byte[] CF = "cf".getBytes();
public static final byte[] ATTR = "attr".getBytes();
...
Get get = new Get(Bytes.toBytes("row1"));
get.setMaxVersions(3); // will return last 3 versions of row
Result r = table.get(get);
byte[] b = r.getValue(CF, ATTR); // returns current version of value
List<Cell> cells = r.getColumnCells(CF, ATTR); // returns all versions of this column
```

Versions

- Implicit Version Example
  - The following Put will be implicitly versioned by HBase with the current time.

```
public static final byte[] CF = "cf".getBytes();
public static final byte[] ATTR = "attr".getBytes();
...
Put put = new Put(Bytes.toBytes(row));
put.add(CF, ATTR, Bytes.toBytes(data));
table.put(put);
```

- Explicit Version Example
  - The following Put has the version timestamp explicitly set.

```
public static final byte[] CF = "cf".getBytes();
public static final byte[] ATTR = "attr".getBytes();
...
Put put = new Put( Bytes.toBytes(row));
long explicitTimeInMs = 555; // just an example
put.add(CF, ATTR, explicitTimeInMs, Bytes.toBytes(data));
table.put(put);
```

HBase and Schema Design

- Schema Creation
  - HBase schemas can be created or updated using the The Apache HBase Shell or by using Admin in the Java API.
  - Tables must be disabled when making ColumnFamily modifications, for example:

```
Configuration config = HBaseConfiguration.create();
Admin admin = new Admin(conf);
TableName table = TableName.valueOf("myTable");
admin.disableTable(table);
HColumnDescriptor cf1 = ...;
admin.addColumn(table, cf1); // adding new ColumnFamily
HColumnDescriptor cf2 = ...;
admin.modifyColumn(table, cf2); // modifying existing ColumnFamily
admin.enableTable(table);
```

Table Schema Rules Of Thumb

- There are many different data sets, with different access patterns and service-level expectations.
  - Aim to have regions sized between 10 and 50 GB.
  - Aim to have cells no larger than 10 MB, or 50 MB if you use mob. Otherwise, consider storing your cell data in HDFS and store a pointer to the data in HBase.
  - A typical schema has between 1 and 3 column families per table. HBase tables should not be designed to mimic RDBMS tables.
  - Around 50-100 regions is a good number for a table with 1 or 2 column families. Remember that a region is a contiguous segment of a column family.
  - Keep your column family names as short as possible. The column family names are stored for every value (ignoring prefix encoding). They should not be self-documenting and descriptive like in a typical RDBMS.

Table Schema Rules Of Thumb

- There are many different data sets, with different access patterns and service-level expectations.
  - If you are storing time-based machine data or logging information, and the row key is based on device ID or service ID plus time, you can end up with a pattern where older data regions never have additional writes beyond a certain age. In this type of situation, you end up with a small number of active regions and a large number of older regions which have no new writes. For these situations, you can tolerate a larger number of regions because your resource consumption is driven by the active regions only.
  - If only one column family is busy with writes, only that column family accumulates memory. Be aware of write patterns when allocating resources.

HBase Versus RDBMS

- HBase is a distributed, column-oriented data storage system.
  - The table schemas mirror the physical storage, creating a system for efficient data structure serialization, storage, and retrieval.
  - The burden is on the application developer to make use of this storage and retrieval in the right way.

- Typical RDBMSs are
  - fixed-schema, row-oriented databases with ACID properties and a sophisticated SQL query engine.
  - The emphasis is on strong consistency, referential integrity, abstraction from the physical layer, and complex queries through the SQL language.
  - You can easily create secondary indexes, perform complex inner and outer joins, count, sum, sort, group, and page your data across a number of tables, rows, and columns.

HBase Versus RDBMS

- Here is a synopsis of how the typical RDBMS scaling story runs. The following list presumes a successful growing service:
  - Initial public launch
    - Move from local workstation to shared, remote hosted MySQL instance with a well-defined schema.
  - Service becomes more popular; too many reads hitting the database
    - Add memcached to cache common queries. Reads are now no longer strictly ACID; cached data must expire.
  - Service continues to grow in popularity; too many writes hitting the database
    - Scale MySQL vertically by buying a beefed up server with 16 cores, 128 GB of RAM, and banks of 15 k RPM hard drives. Costly.
  - New features increases query complexity; now we have too many joins
    - Denormalize your data to reduce joins. (That’s not what they taught me in DBA school!)
  - Rising popularity swamps the server; things are too slow
    - Stop doing any server-side computations.
  - Some queries are still too slow
    - Periodically prematerialize the most complex queries, try to stop joining in most cases.
  - Reads are OK, but writes are getting slower and slower
    - Drop secondary indexes and triggers (no indexes?).

HBase Versus RDBMS

- Enter HBase, which has the following characteristics:
  - No real indexes
    - Rows are stored sequentially, as are the columns within each row. Therefore, no issues with index bloat, and insert performance is independent of table size.
  - Automatic partitioning
    - As your tables grow, they will automatically be split into regions and distributed across all available nodes.
  - Scale linearly and automatically with new nodes
    - Add a node, point it to the existing cluster, and run the region server. Regions will automatically rebalance and load will spread evenly.
  - Commodity hardware
    - Clusters are built on $1,000–$5,000 nodes rather than $50,000 nodes. RDBMSs are I/O hungry, requiring more costly hardware.
  - Fault tolerance
    - Lots of nodes means each is relatively insignificant. No need to worry about individual node downtime.
  - Batch processing
    - MapReduce integration allows fully parallel, distributed jobs against your data with locality awareness.

References

- Hadoop: The Definitive Guide
  - By Tom White
  - O'Reilly Publishing
- Apache HBase
  - https://hbase.apache.org
- Apache HBase ™ Reference Guide
  - https://hbase.apache.org/book.html#quickstart
- hbase通过idea操作api
  - https://blog.csdn.net/u010800708/article/details/86742516
- Introduction to Basic Schema Design by Amandeep Khurana
  - http://0b4af6cdc2f0c5998459-c0245c5c937c5dedcca3f1764ecc9b2f.r43.cf2.rackcdn.com/9353-login1210_khurana.pdf

REin

Thank You!