HBase Architecture

REAL-TIME READING & WRITING
Column-Oriented Databases

- Data layout
  - data saved is grouped by column
  - subsequent column values are stored contiguously on disk
  - RDBMS store data by row

- Specialized databases for specific workloads:
  - reduced I/O
  - better suited for compression
    - efficient use of bandwidth
  - column values are often very similar and differ little
  - row-by-row real-time access to data
Column-Oriented Databases

Important Note:

- HBase is not a column-oriented DB
- storage is via HFile
- access via row key (column family, column ...)
- provides key-based access to specific cell of data
- a sequential range of cells
# Column-Oriented

**Row Oriented (RDBMS Model)**

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>Age</th>
<th>Interests</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ricky</td>
<td></td>
<td>Soccer, Movies, Baseball</td>
</tr>
<tr>
<td>2</td>
<td>Ankur</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Sam</td>
<td>25</td>
<td>Music</td>
</tr>
</tbody>
</table>

**Column Oriented (Multi-value sorted map)**

<table>
<thead>
<tr>
<th>id</th>
<th>Name</th>
<th>id</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ricky</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Ankur</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>Sam</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>Interests</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Soccer</td>
</tr>
<tr>
<td>1</td>
<td>Movies</td>
</tr>
<tr>
<td>1</td>
<td>Baseball</td>
</tr>
<tr>
<td>3</td>
<td>Music</td>
</tr>
</tbody>
</table>
Non-Relational Data Stores

- **NoSQL = not only SQL**
  - In practice, this is becoming a thin line to make the distinction
    - one difference is in the data model
    - another difference is in the consistency

- **Consistency models**
  - **Strict**
    - all changes to data are atomic
  - **Sequential**
    - changes to data are seen in the same order as they were applied
  - **Eventual**
    - updates propagate through the system
    - replicas are completed
    - this is the steady state
HBase Terms

- **Node**
  - a single computer

- **Cluster**
  - a group of nodes connected and coordinated by certain nodes to perform tasks

- **Master Node**
  - a node performing coordination tasks

- **Worker Node**
  - a node performing tasks assigned to it by a master node

- **Daemon**
  - a process or program that runs in the background
About HBase Tables

- HBase is essentially a distributed, sorted map
- Distributed:
  - HBase is designed to use multiple machines to store and serve table data
- Sorted Map:
  - HBase stores table data as a map
  - guarantees that adjacent keys will be stored next to each other on disk
Example Application Data

- In our examples, we will use a table that holds:
  - User contact information
  - Profile photos
  - User sign-in information such as username and password
  - Settings or preferences for multiple applications

- The table will be designed to provide access to the data based on the username
Example Application Data

- Not every field will have a value
  - The application that stores data to a given field has not run
  - Alternatively, the user may have elected not to provide all the information

- For now we will focus on the contact information and profile photo
  - Contact information
    - First Name
    - Last Name
  - Profile photo
    - Image that the user uploads
## Example Data

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>sperberp</td>
<td>fname: Phyllis Iname: Sperber</td>
</tr>
<tr>
<td>turnerb</td>
<td>fname: Brian Iname: Turner</td>
</tr>
<tr>
<td>walkerm</td>
<td>fname: Martin Iname: Walker</td>
</tr>
<tr>
<td>gurbaxanir</td>
<td>fname: Ravi Iname: Gurbaxani</td>
</tr>
<tr>
<td>millerj</td>
<td>fname: John Iname: Miller</td>
</tr>
<tr>
<td>nunezm</td>
<td>fname: Misty Iname: Nunez</td>
</tr>
<tr>
<td>rossw</td>
<td>fname: Wallace Iname: Ross</td>
</tr>
<tr>
<td>jordana</td>
<td>fname: Jordan Iname: Astros</td>
</tr>
<tr>
<td>prashatz</td>
<td>fname: Prashat Iname: Zifar</td>
</tr>
<tr>
<td>avabills</td>
<td>fname: Avabill Iname: Suki</td>
</tr>
<tr>
<td>harrisonw</td>
<td>fname: Harrison Iname: Williams</td>
</tr>
</tbody>
</table>
HBase Architecture

- Regions are the basic element of availability and distribution for tables

- Master Node: HMaster
  - Assigns Regions to Region Servers
  - via ZooKeeper
  - Handles load balancing
  - Not part of the data path
    - works with metadata and schemas only

- Region Servers
  - Handle reads and writes
  - Handle region splitting
Daemons in an HBase Cluster

- **RegionServer**
  - responsible for serving and managing regions

- **Master**
  - monitors all RegionServer instances in the cluster
  - interface for all metadata changes

- **ZooKeeper**
  - a centralized service used to maintain configuration information for HBase

- **NameNode**
  - keeps track of HDFS metadata

- **DataNode**
  - keeps track of HDFS blocks
Regions

- Tables are broken into smaller pieces called regions.
- A region contains a series of rows spanning from the Start row key to the end row key defined for that region.
- A region is served by a RegionServer.

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>sperberp</td>
<td>fname: Phyllis  lname: Sperber</td>
</tr>
<tr>
<td>turnerb</td>
<td>fname: Brian  lname: Turner</td>
</tr>
<tr>
<td>walkerm</td>
<td>fname: Martin lname: Walker</td>
</tr>
<tr>
<td>gurbaxanir</td>
<td>fname: Ravi  lname: Gurbaxani</td>
</tr>
<tr>
<td>millerj</td>
<td>fname: John   lname: Miller</td>
</tr>
<tr>
<td>nunezm</td>
<td>fname: Misty lname: Nunez</td>
</tr>
<tr>
<td>rossw</td>
<td>fname: Wallace lname: Ross</td>
</tr>
<tr>
<td>jordana</td>
<td>fname: Jordan lname: Astros</td>
</tr>
<tr>
<td>prashatz</td>
<td>fname: Prashat lname: Zifar</td>
</tr>
<tr>
<td>avabills</td>
<td>fname: Avabill lname: Suki</td>
</tr>
<tr>
<td>harrisonw</td>
<td>fname: Harrison lname: Williams</td>
</tr>
</tbody>
</table>
### Users Table Divided into Regions

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table – Region 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>avabill</td>
<td>fname: Avabill  lname: Suki</td>
</tr>
<tr>
<td>harrisonw</td>
<td>fname: Brian  lname: Turner</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table – Region 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>jordana</td>
<td>fname: Jordan  lname: Astros</td>
</tr>
<tr>
<td>prashat</td>
<td>fname: Brian  lname: Turner</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table – Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>millerj</td>
<td>fname: John  lname: Miller</td>
</tr>
<tr>
<td>nunezm</td>
<td>fname: Misty  lname: Nunez</td>
</tr>
<tr>
<td>rossw</td>
<td>fname: Wallace  lname: Ross</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table – Region 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>sperberp</td>
<td>fname: Phyllis  lname: Sperber</td>
</tr>
<tr>
<td>turnerb</td>
<td>fname: Brian  lname: Turner</td>
</tr>
<tr>
<td>walkerm</td>
<td>fname: Martin  lname: Walker</td>
</tr>
<tr>
<td>gurbaxanir</td>
<td>fname: Ravi  lname: Gurbaxani</td>
</tr>
</tbody>
</table>
Regions Served by RegionServers

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table – Region 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>avabill</td>
<td></td>
</tr>
<tr>
<td>harrisonw</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table – Region 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>sperberp</td>
<td>fname: Phyllis Iname: Sperber</td>
</tr>
<tr>
<td>turnerb</td>
<td>fname: Brian Iname: Turner</td>
</tr>
<tr>
<td>walkerm</td>
<td>fname: Martin Iname: Walker</td>
</tr>
<tr>
<td>gurbaxanir</td>
<td>fname: Ravi Iname: Gurbaxani</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table – Region 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>jordana</td>
<td></td>
</tr>
<tr>
<td>prashat</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Users Table – Region 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>millerj</td>
<td>fname: John Iname: Miller</td>
</tr>
<tr>
<td>nunezm</td>
<td>fname: Misty Iname: Nunez</td>
</tr>
<tr>
<td>rossw</td>
<td>fname: Wallace Iname: Ross</td>
</tr>
</tbody>
</table>
Placement of Daemons and Services

Master Nodes
- Master 1
  - NameNode
  - Master
  - Zookeeper
- Master 2
  - Standby NameNode
  - Master
  - Zookeeper
- Master 3
  - Master
  - Zookeeper

Worker Nodes
- Worker 1
  - RegionServer
  - DataNode
- Worker 2
  - RegionServer
  - DataNode
- Worker 3
  - RegionServer
  - DataNode
- Worker 4
  - RegionServer
  - DataNode
- Worker n
  - RegionServer
  - DataNode
Region Servers

- Handle all read and write requests
  - for all regions they serve
- Will split regions that have exceeded the region size threshold
- Clients communicate directly with RS for data-related operations
- Can be added or removed while the system is running
Regions

- Vehicle for scalability and load-balancing of data
- Equivalent of range partitions in sharded RDBMS
- Dynamically split by the system
  - when they become too large
  - can also be merged to reduce the number of Storage Files
HMaster

- Responsible for assigning regions to region servers
- Handles metadata operations
  - schema changes
  - creation of tables and column families
- Handles load balancing of regions across region servers
- Uses ZooKeeper coordination service
- Maintains the configured state of the HBase cluster
- Does not provide any data services
HBase Daemon Architecture

Legend:
- An HBase RegionServer is collocated with an HDFS DataNode.
- HBase clients communicate directly with Region Servers for sending and receiving data.
- HMaster manages Region assignment and handles DDL operations.
- Online configuration state is maintained in ZooKeeper.
- HMaster and ZooKeeper are NOT involved in data path.

Regions (continued)

- Contain a Store for each Column Family defined in the Region
  - Stores contain
    - StoreFiles which are sequences of blocks configured on a column family
      - Default block size is 64K (configurable)
    - StoreFiles contain HFiles
      - HFiles are the unit of transport between the RegionServer and the HDFS

- Owned by a specific Region Server

- Initially 1 region per table (connection to a Region Server)

- Each region size is configured
  - the number of regions per Region Server is configured
  - HMaster monitors region size: if a threshold is attained region is split
  - regions are split in two at the middle key
  - this creates roughly two equivalent (in size) regions
Table to Region to Region Server Mapping

Logical Architecture
Distributed, persistent partitions of a BigTable

Legend:
- A single table is partitioned into Regions of roughly equal size.
- Regions are assigned to Region Servers across the cluster.
- Region Servers host roughly the same number of regions.
Logical Data Model

A sparse, multi-dimensional, sorted map

<table>
<thead>
<tr>
<th>Rowkey</th>
<th>Column Family</th>
<th>Column Qualifier</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>cf1</td>
<td>&quot;bar&quot;</td>
<td>1368394583</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1368394261</td>
<td>&quot;hello&quot;</td>
</tr>
<tr>
<td>b</td>
<td>cf2</td>
<td>&quot;2011-07-04&quot;</td>
<td>1368396302</td>
<td>&quot;fourth of July&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1.0001</td>
<td>1368387684</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;thumb&quot;</td>
<td>1368387247</td>
<td>[3.6 kb png data]</td>
</tr>
</tbody>
</table>

Legend:
- Rows are sorted by rowkey.
- Within a row, values are located by column family and qualifier.
- Values carry a timestamp; there can be multiple versions of a value.
- Within a column family, data is schemaless. Qualifiers and values are treated as arbitrary bytes.
HFile

- Holds sorted key/value pairs, ordered by key
- StoreFiles are sequences of blocks configured column families
  - default block size is 64K (configurable)
- Index stored at the end
  - index is loaded when the HFile is opened and kept in memory
- Advantages
  - lookups performed with single disk seek
  - keys found through an in-memory search of the block index
    - followed by a block read from disk to find the actual key
Disk Storage

HFile 232MB

HDFS File

Block 1: 128MB

Block 2: 104MB

image: http://ofps.oreilly.com/titles/9781449396107/architecture.html
Write-Ahead Log

- Eventual consistency model
- Ensures resiliency to failures
- Region servers keep data in-memory until flush
- In case of crashes or power loss
  - WAL is a common approach to address fault-tolerance
  - Every data update is first written to a log
- Log is persisted
  - And *replicated* - resides in HDFS
- Client is notified a successful operation on data
Write-Ahead Log
HBase Components
Data Model

- **Table**
  - applications store data into an HBase table
  - tables are made of rows and columns

- **Cells**
  - intersection of row and column coordinates
  - versioned (multidimensional – data is immutable)
  - a \{row, column, version\} tuple

- **Versioning**
  - 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
  - specified as a long integer
  - is stored in decreasing order
    - so when reading from a store file, the most recent values are found first
Rows

Figure 1-5. A time-oriented view into parts of a row

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Time Stamp</th>
<th>Column “data:”</th>
<th>Column “meta:”</th>
<th>Column “counters:”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“row1”</td>
<td>t3</td>
<td>{&quot;name&quot;:&quot;lars&quot;,&quot;address&quot;:...}</td>
<td>“2323”</td>
<td>“1”</td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td>{&quot;name&quot;:&quot;lars&quot;,&quot;address&quot;:...}</td>
<td></td>
<td>“2”</td>
</tr>
<tr>
<td>t8</td>
<td></td>
<td></td>
<td>“application/json”</td>
<td></td>
</tr>
<tr>
<td>t9</td>
<td></td>
<td>{&quot;name&quot;:&quot;lars&quot;,&quot;address&quot;:...}</td>
<td></td>
<td>“3”</td>
</tr>
</tbody>
</table>

Figure 1-6. The same parts of the row rendered as a spreadsheet
Row Keys

- **Row Key**
  - Table row keys are also byte arrays
  - Almost anything can serve as a row key
    - From strings to binary representations of longs or even serialized data structures

- **Lexicographically sorted**
  - With the lowest order appearing first in a table
  - The empty byte array is used to denote both the start and end of a table’s namespace
  - All table accesses are via the table row key — its primary key
Columns

- Each column may have multiple versions
- Each distinct value contained in a separate cell
  - one or more columns form a row
  - addressed uniquely by row key

```
hbase(main):001:0> scan 'table1'
ROW COLUMN+CELL
row-1 column=cf1:, timestamp=1297073325971 ...
row-10 column=cf1:, timestamp=1297073337383 ...
row-11 column=cf1:, timestamp=1297073340493 ...
row-2 column=cf1:, timestamp=1297073329851 ...
row-22 column=cf1:, timestamp=1297073344482 ...
row-3 column=cf1:, timestamp=129707333504 ...
row-abc column=cf1:, timestamp=1297073349875 ...
7 row(s) in 0.1100 seconds
```
Cells

- Sparse storage and NULL values
  - in RDBMS NULL cells need to be set and occupy space
  - in HBase, NULL cells or columns are simply not stored

- Cell
  - every value is timestamped
  - multiple versions of a value that changes over time can be saved
  - versions are stored in decreasing timestamp, most recent first
  - cell versions can be constrained by predicate deletions
    - example: Keep only values from the last week
Column Families

- Columns grouped into column families
- All column members of a column family have the same prefix
  - Examples of members of the same column family:
    - courses:history
    - courses:math
- All column family members stored together in the filesystem
- Should have the same general access pattern and size characteristics
- Tuning done at the column family level
**Column Families and Columns**

## Sorted Map Datastore
*(physical view as “cells”)*

### Info Column Family / Locality Group

<table>
<thead>
<tr>
<th>Row key</th>
<th>Column key</th>
<th>Timestamp</th>
<th>Cell value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cutting</td>
<td>info:height</td>
<td>1273516197868</td>
<td>9ft</td>
</tr>
<tr>
<td>cutting</td>
<td>info:state</td>
<td>1043871824184</td>
<td>CA</td>
</tr>
<tr>
<td>tipcon</td>
<td>info:height</td>
<td>1273878447049</td>
<td>5ft7</td>
</tr>
<tr>
<td>tipcon</td>
<td>info:state</td>
<td>1273616297446</td>
<td>CA</td>
</tr>
</tbody>
</table>

### Roles Column Family / Locality Group

<table>
<thead>
<tr>
<th>Row key</th>
<th>Column key</th>
<th>Timestamp</th>
<th>Cell value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cutting</td>
<td>roles:ASF</td>
<td>1273871823022</td>
<td>Director</td>
</tr>
<tr>
<td>cutting</td>
<td>roles:Hadoop</td>
<td>1183746289103</td>
<td>Founder</td>
</tr>
<tr>
<td>tipcon</td>
<td>roles:Hadoop</td>
<td>1300062064923</td>
<td>PMC</td>
</tr>
<tr>
<td>tipcon</td>
<td>roles:Hadoop</td>
<td>1293388212294</td>
<td>Committer</td>
</tr>
<tr>
<td>tipcon</td>
<td>roles:Hive</td>
<td>1273616297446</td>
<td>Contributor</td>
</tr>
</tbody>
</table>

*Sorted on disk by Row key, Column key, descending timestamp*

*Milliseconds since unix epoch*
## Data Storage Within a Column Family

<table>
<thead>
<tr>
<th>Row Key</th>
<th>Column</th>
<th>Timestamp</th>
<th>Cell Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>jdupont</td>
<td>contactinfo:fname</td>
<td>1273746289103</td>
<td>Dupont</td>
</tr>
<tr>
<td>jdupont</td>
<td>contactinfo:lname</td>
<td>1273878447049</td>
<td>Mary</td>
</tr>
<tr>
<td>Jsmith</td>
<td>contactinfo:fname</td>
<td>1273516197868</td>
<td>Jacob</td>
</tr>
<tr>
<td>Jsmith</td>
<td>contactinfo:lname</td>
<td>1273871824184</td>
<td>Ravi</td>
</tr>
<tr>
<td>mlabel</td>
<td>contactinfo:fname</td>
<td>1273616297446</td>
<td>Susan</td>
</tr>
<tr>
<td>mlabel</td>
<td>contactinfo:lname</td>
<td>1273971921442</td>
<td>Raputh</td>
</tr>
</tbody>
</table>
HBase Operations

- All rows in HBase are identified by a row key
  - this is like the primary key in a relational database

- Get/Scan retrieves data
  - a Get retrieves a single row using the row key
  - a Scan retrieves all rows
  - a Scan can be constrained to retrieve all rows between a Start row key and an end row key

- Put inserts data
  - a Put adds a new row identified by a row key
  - multiple Put calls can be run to insert multiple rows with different row keys
HBase Operations

- **Delete** marks data as having been deleted
  - a Delete removes the row identified by a row key
  - the data is not removed from HDFS during the call but is marked for deletion
  - physical deletion from HDFS happens later

- **Increment** allows atomic counters
  - cells containing a value stored as a 64-bit integer (a long)
  - increment allows the value to be initially set, or incremented if it already has a value
  - atomicity allows for concurrent access from multiple clients without fear of corruption by a write from another process
Row Key is the Only Indexed Column

- RDBMSs can have as many index columns as required
- In HBase, we have just one indexed column — the row key
- Significant effort goes into the row key planning for HBase tables
- We rely on the row key to provide quick access to data for all applications that use a given table
## Features Comparison: RDBMS vs. HBase

<table>
<thead>
<tr>
<th></th>
<th>RDBMS</th>
<th>HBase</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data layout</strong></td>
<td>Row- or column-oriented</td>
<td>Column family-oriented</td>
</tr>
<tr>
<td><strong>Transactions</strong></td>
<td>Yes</td>
<td>Single row only</td>
</tr>
<tr>
<td><strong>Query language</strong></td>
<td>SQL</td>
<td>get/ put/scan</td>
</tr>
<tr>
<td><strong>Security</strong></td>
<td>Authentication/Authorization</td>
<td>Access control at per-cell level, also at</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cluster, table, or row level</td>
</tr>
<tr>
<td><strong>Indexes</strong></td>
<td>Yes</td>
<td>Row key only</td>
</tr>
<tr>
<td><strong>Max data Size</strong></td>
<td>TBs</td>
<td>PBs</td>
</tr>
<tr>
<td><strong>Read/write throughput limits</strong></td>
<td>1000s queries/second</td>
<td>Millions Of queries/second</td>
</tr>
</tbody>
</table>
Comparison with RDBMS Table Design

- **Joins**
  - in relational databases, one would typically normalize tables and use joins to retrieve data
  - HBase does not support explicit joins
  - instead, a lookup by row key joins data from column families

- **Scaling**
  - relational tables can scale through partitioning or sharding data
  - HBase automatically partitions data into smaller pieces
HBase vs. RDBMS Schema Design

- Relationship-centric for designing an RDBMS schema
  - determine all the types of data to be stored
  - determine relationships between data elements
  - create tables, columns, and foreign keys to maintain relationships

- Data-centric for designing an HBase schema
  - identify ways in which data will be accessed
  - identify types of data to be stored
  - create data layouts and keys
HBase Schemas

- Column families
  - Defined at table-creation time
  - It’s possible to alter a table and add new column families
  - Table must be disabled at altering time

- Changes (e.g., region size, block size)
  - Take effect during major compaction
  - StoreFiles get re-written
Lab
Zookeeper
HBase and ZooKeeper

- An HBase cluster can have multiple Masters for high availability
  - only one Master controls the cluster
  - the ZooKeeper service handles coordination of the Masters

- The ZooKeeper service runs on master nodes on the cluster
  - upon startup all Masters connect to ZooKeeper
  - they compete to run the cluster
  - the first Master to connect "wins" control
  - if the controlling Master fails
  - the remaining Masters will compete again to run the cluster
Table Types

- **Userspace tables**
  - HBase tables created with the HBase API or HBase shell
  - the Users table in the previous section is a userspace table

- **The catalog table, hbase:meta**
  - special table used and accessed Only by HBase
  - keeps track Of the locations Of RegionServers and regions
  - hbase:meta is an HBase table but it is filtered out of the HBase shell's list command

- **Location of the hbase:meta table** is found through a lookup in ZooKeeper
The `hbase:meta` Table

- The first query from a client is to ZooKeeper to find the location of `hbase:meta`.
- The second query is to `hbase:meta`.
- `hbase:meta` lists all regions and their locations.
- The `hbase:meta` table is never split into regions.
- The third query is to the RegionServer where the region for the data is held.
- Results of the first two queries are cached by the client.
Querying for Regions

1. Client
2. RegionServer 1
   - hbase:meta
   - Table 1 – Region 1
   - Table 1 – Region 4
3. RegionServer 2
   - Table 1 – Region 2
   - Table 1 – Region 3
4. Zookeeper
Monitoring the Cluster with Hadoop/HBase Web UIs

- All Hadoop and HBase daemons contain a Web server
  - exposes information over a well-known port
  - the type of information and content are specific to the daemon

- Some important ones for HBase are:
  - HBase Master `http://[master_address]: 60010`
  - RegionServer `http://[regionserver_address]: 60030`

- Looking at the NameNode is useful for HBase
  - NameNode `http://[namenode_address]: 50070`
Diagnosing Problems Using Log Files

- All log files are written to `/var/log/base` by default
- Log files can be viewed using the daemons' web interfaces
- The Web interfaces can be used to dynamically set the logging level
- Many issues can be diagnosed using the logs
- Thrift and REST errors are only logged in the log file and are not always sent to the client
HBase and HDFS

- The HBase RegionServer writes data to HDFS on its local disk
- HDFS will replicate the data to other nodes in the cluster
- Replication ensures the data remains available even if a node fails

![Diagram of HBase and HDFS components](image-url)
Lab
End of Chapter