Architecture of Enterprise Applications 22
HBase & Hive

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

Agenda

- Hive
  - An Example
  - Comparison with traditional db
  - Hive QL
  - Tables
  - Query
- HBase
  - Tables
  - Clients
  - HBase vs. RDBMS

Hive

- Hive, a framework for data warehousing on top of Hadoop.
  - Hive grew from a need to manage and learn from the huge volumes of data that Facebook was producing every day from its burgeoning social network.
  - After trying a few different systems, the team chose Hadoop for storage and processing, since it was cost-effective and met their scalability needs.
- Hive was created to make it possible for analysts with strong SQL skills (but meager Java programming skills) to run queries on the huge volumes of data that Facebook stored in HDFS.
  - Today, Hive is a successful Apache project used by many organizations as a general-purpose, scalable data processing platform.

Hive

- In normal use, Hive runs on your workstation and converts your SQL query into a series of MapReduce jobs for execution on a Hadoop cluster.
  - Hive organizes data into tables, which provide a means for attaching structure to data stored in HDFS.
  - Metadata— such as table schemas—is stored in a database called the metastore.

An Example

REliable, INtelligent & Scalable Systems

- Let's see how to use Hive to run a query on the weather dataset.
- The first step is to load the data into Hive's managed storage.
- Just like an RDBMS, Hive organizes its data into tables. We create a table to hold the weather data using the CREATE TABLE statement:

```
CREATE TABLE records
        (year STRING, temperature INT, quality INT)
ROW FORMAT DELIMITED
   FIELDS TERMINATED BY '\t';
```

An Example

- Next we can populate Hive with the data.
  - This is just a small sample, for exploratory purposes:

```
LOAD DATA LOCAL INPATH 'input/ncdc/micro-tab/sample.txt'
OVERWRITE INTO TABLE records;
```

- Running this command tells Hive to put the specified local file in its warehouse directory.
  - There is no attempt, for example, to parse the file and store it in an internal database format, since Hive does not mandate any particular file format.
  - Files are stored verbatim: they are not modified by Hive.

An Example

- In this example, we are storing Hive tables on the local filesystem (fs.default.name is set to its default value of file:///).
  - Tables are stored as directories under Hive's warehouse directory, which is controlled by the hive.metastore.warehouse.dir, and defaults to /user/hive/warehouse.
  - Thus, the files for the records table are found in the /user/hive/warehouse/records directory on the local filesystem:

    % ls /user/hive/warehouse/record/sample.txt

- In this case, there is only one file, sample.txt, but in general there can be more, and Hive will read all of them when querying the table.

An Example

- Now that the data is in Hive, we can run a query against it:

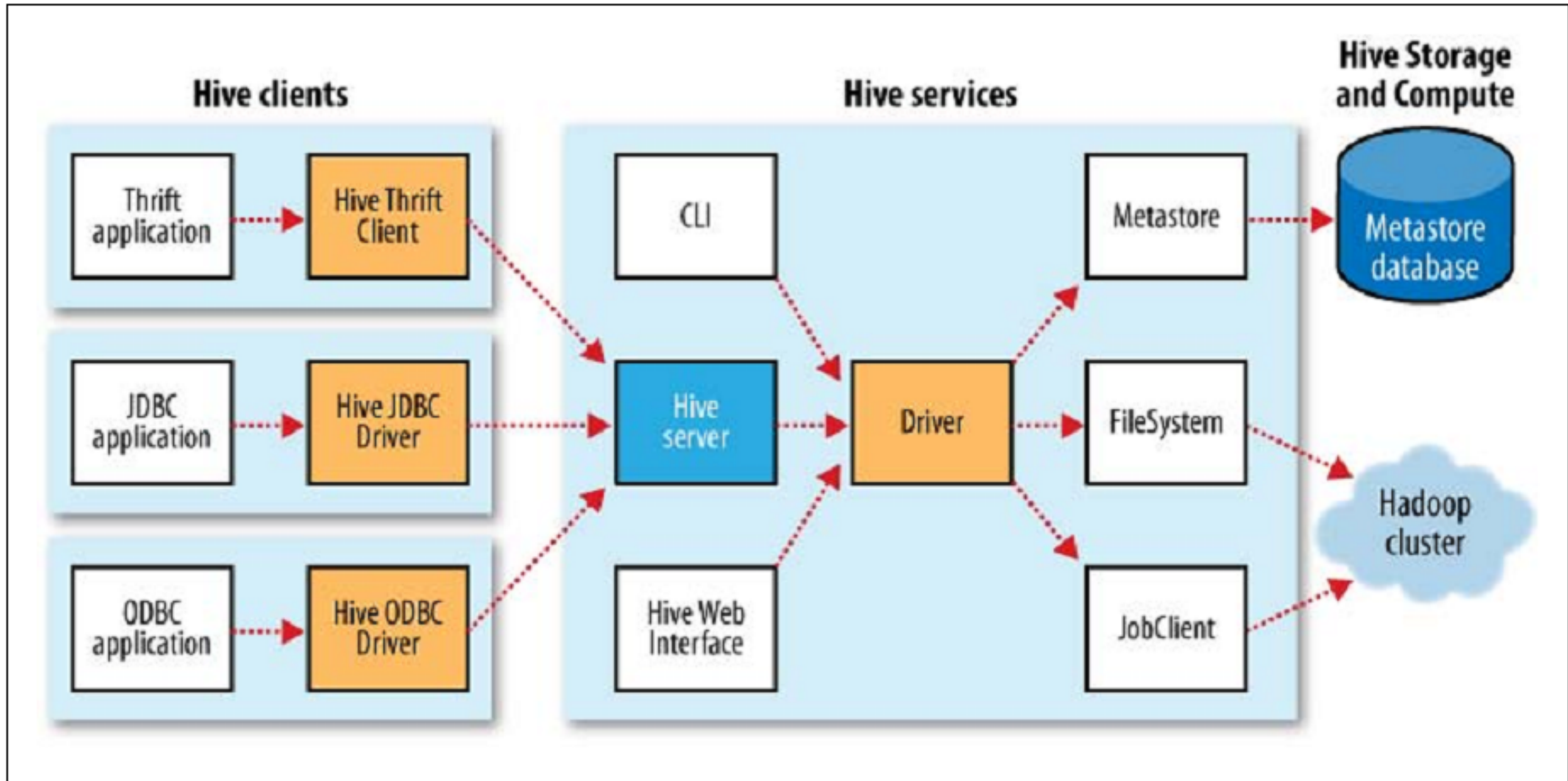
```
hive> SELECT year, MAX(temperature)
    > FROM records
    > WHERE temperature != 9999
    > AND (quality = 0 OR quality = 1 OR quality = 4 OR quality = 5 OR quality = 9)
    > GROUP BY year;
1949 111
1950 22
```

Hive Services

REliable, INtelligent & Scalable Systems

- The Hive shell is only one of several services that you can run using the hive command.
- Type hive –service help to get a list of available service names; the most useful are described below.
  - cli
    - The command line interface to Hive (the shell). This is the default service.
  - hiveserver
    - Runs Hive as a server exposing a Thrift service, enabling access from a range of clients written in different languages.
  - hwi
    - The Hive Web Interface.
  - jar
    - The Hive equivalent to hadoop jar, a convenient way to run Java applications that includes both Hadoop and Hive classes on the classpath.
  - metastore
    - By default, the metastore is run in the same process as the Hive service. Using this service, it is possible to run the metastore as a standalone (remote) process.

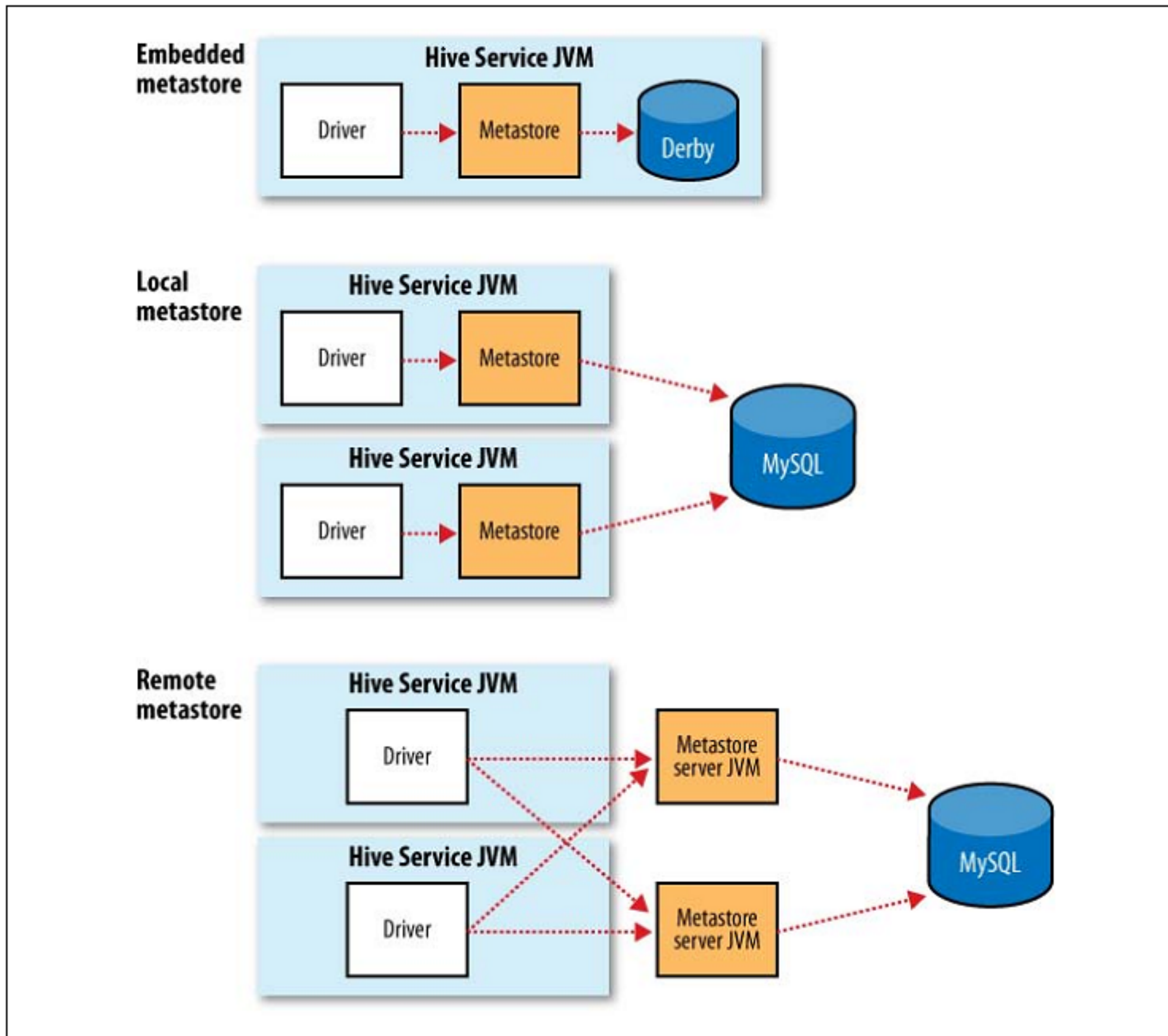
Hive clients



The Metastore

- The metastore is the central repository of Hive metadata.
  - The metastore is divided into two pieces: a service and the backing store for the data.
  - By default, the metastore service runs in the same JVM as the Hive service and contains an embedded Derby database instance backed by the local disk.
  - It also supports multiple sessions (and therefore multiple users) is to use a standalone database. This configuration is referred to as a local metastore, since the metastore service still runs in the same process as the Hive service, but connects to a database running in a separate process, either on the same machine or on a remote machine.
  - Going a step further, there's another metastore configuration called a remote metastore, where one or more metastore servers run in separate processes to the Hive service.

The Metastore



Comparison with Traditional Databases

- Schema on Read Versus Schema on Write
  - In a traditional database, a table's schema is enforced at data load time. If the data being loaded doesn't conform to the schema, then it is rejected.
    - This design is sometimes called schema on write, since the data is checked against the schema when it is written into the database.
  - Hive, on the other hand, doesn't verify the data when it is loaded, but rather when a query is issued.
    - This is called schema on read.
  - There are trade-offs between the two approaches.
    - Schema on read makes for a very fast initial load, since the data does not have to be read, parsed, and serialized to disk in the database's internal format.
    - Schema on write makes query time performance faster, since the database can index columns and perform compression on the data.

HiveQL

- Hive's SQL dialect, called HiveQL, does not support the full SQL-92 specification.

Feature	SQL	HiveQL	References
Updates	UPDATE, INSERT, DELETE	INSERT OVERWRITE TABLE (populates whole table or partition)	"INSERT OVERWRITE TABLE" on page 392, "Updates, Transactions, and Indexes" on page 376
Transactions	Supported	Not supported	
Indexes	Supported	Not supported	
Latency	Sub-second	Minutes	
Data types	Integral, floating point, fixed point, text and binary strings, temporal	Integral, floating point, boolean, string, array, map, struct	"Data Types" on page 378
Functions	Hundreds of built-in functions	Dozens of built-in functions	"Operators and Functions" on page 380
Multitable inserts	Not supported	Supported	"Multitable insert" on page 393
Create table as select	Not valid SQL-92, but found in some databases	Supported	"CREATE TABLE...AS SELECT" on page 394

HiveQL

REliable, INtelligent & Scalable Systems

- Hive's SQL dialect, called HiveQL, does not support the full SQL-92 specification.

Feature	SQL	HiveQL	References
Select	SQL-92	Single table or view in the FROM clause. SORT BY for partial ordering. LIMIT to limit number of rows returned. HAVING not supported.	"Querying Data" on page 395
Joins	SQL-92 or variants (join tables in the FROM clause, join condition in the WHERE clause)	Inner joins, outer joins, semi joins, map joins. SQL-92 syntax, with hinting.	"Joins" on page 397
Subqueries	In any clause. Correlated or noncorrelated.	Only in the FROM clause. Correlated subqueries not supported	"Subqueries" on page 400
Views	Updatable. Materialized or nonmaterialized.	Read-only. Materialized views not supported	"Views" on page 401
Extension points	User-defined functions. Stored procedures.	User-defined functions. MapReduce scripts.	"User-Defined Functions" on page 402, "MapReduce Scripts" on page 396

Tables

- A Hive table is logically made up of the data being stored and the associated metadata describing the layout of the data in the table.
  - The data typically resides in HDFS, although it may reside in any Hadoop filesystem, including the local filesystem or S3.
  - Hive stores the metadata in a relational database—and not in HDFS.

- Managed Tables and External Tables
  - When you create a table in Hive, by default Hive will manage the data, which means that Hive moves the data into its warehouse directory.
  - Alternatively, you may create an external table, which tells Hive to refer to the data that is at an existing location outside the warehouse directory.

Tables

- Partitions and Buckets
  - Hive organizes tables into partitions, a way of dividing a table into coarse-grained parts based on the value of a partition column.
  - Partitions are defined at table creation time using the PARTITIONED BY clause, which takes a list of column definitions.
    - For the hypothetical log files example, we might define a table with records comprising a timestamp and the log line itself:

```
CREATE TABLE logs (ts BIGINT, line STRING)
PARTITIONED BY (dt STRING, country STRING);
```

  - When we load data into a partitioned table, the partition values are specified explicitly:

```
LOAD DATA LOCAL INPATH 'input/hive/partitions/file1'
INTO TABLE logs
PARTITION (dt='2001-01-01', country='GB');
```

Tables

REliable, INtelligent & Scalable Systems

- Partitions and Buckets
  - Tables or partitions may further be subdivided into buckets, to give extra structure to the data that may be used for more efficient queries.
  - There are two reasons why you might want to organize your tables (or partitions) into buckets.
    - The first is to enable more efficient queries. Bucketing imposes extra structure on the table, which Hive can take advantage of when performing certain queries.
    - The second reason to bucket a table is to make sampling more efficient. When working with large datasets, it is very convenient to try out queries on a fraction of your dataset while you are in the process of developing or refining them.

```
CREATE TABLE bucketed_users (id INT, name STRING)
CLUSTERED BY (id) SORTED BY (id ASC)INTO 4 BUCKETS;
```

Querying Data

- Sorting and Aggregating

```
hive> FROM records2
    > SELECT year, temperature
    > DISTRIBUTE BY year
    > SORT BY year ASC, temperature DESC;
1949 111
1949 78
1950 22
1950 0
1950 -11
```

Querying Data

- MapReduce Scripts
  - Using an approach like Hadoop Streaming, the TRANSFORM, MAP, and REDUCE clauses make it possible to invoke an external script or program from Hive.

```
FROM (
  FROM records2
  MAP year, temperature, quality
  USING 'is_good_quality.py'
  AS year, temperature) map_output
REDUCE year, temperature
USING 'max_temperature_reduce.py'
AS year, temperature;
```

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.

HBase

REliable, INtelligent & Scalable Systems

- 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.

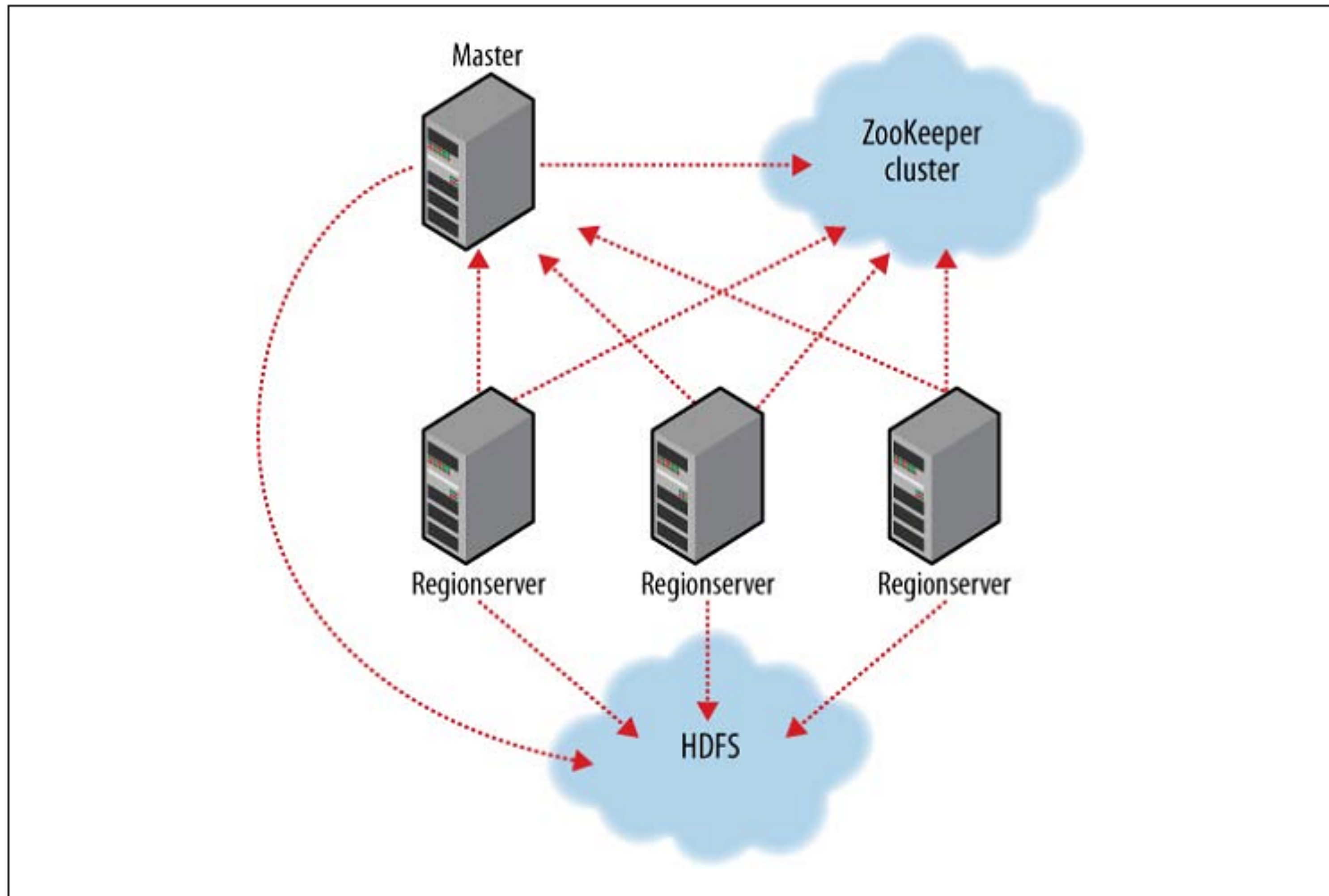
HBase

- 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.

REin
REliable, INtelligent & Scalable Systems

- 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.

HBase cluster members



HBase Clients

REliable, INtelligent & Scalable Systems

- HBase, like Hadoop, is written in Java.

```
public class ExampleClient {
    public static void main(String[] args) throws IOException {
        Configuration config = HBaseConfiguration.create();

        // Create table
        HBaseAdmin admin = new HBaseAdmin(config);
        HTableDescriptor htd = new HTableDescriptor("test");
        HColumnDescriptor hcd = new HColumnDescriptor("data");
        htd.addFamily(hcd);
        admin.createTable(htd);
        byte [] tablename = htd.getName();
        HTableDescriptor [] tables = admin.listTables();
        if (tables.length != 1 && Bytes.equals(tablename, tables[0].getName())) {
            throw new IOException("Failed create of table");
        }
```

HBase Clients

REliable, INtelligent & Scalable Systems

- HBase, like Hadoop, is written in Java.

```
// Run some operations -- a put, a get, and a scan -- against the table.
HTable table = new HTable(config, tablename);
byte [] row1 = Bytes.toBytes("row1");
Put p1 = new Put(row1);
byte [] databytes = Bytes.toBytes("data");
p1.add(databytes, Bytes.toBytes("1"), Bytes.toBytes("value1"));
table.put(p1);
Get g = new Get(row1);
Result result = table.get(g);
System.out.println("Get: " + result);
Scan scan = new Scan();
ResultScanner scanner = table.getScanner(scan);
try {
  for (Result scannerResult: scanner) {
        System.out.println("Scan: " + scannerResult);
}
} finally {
  scanner.close();
}
```

HBase Clients

- HBase, like Hadoop, is written in Java.

```
        // Drop the table
        admin.disableTable(tablename);
        admin.deleteTable(tablename);
    }
}
```

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

REliable, INtelligent & Scalable Systems

- 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

REliable, INtelligent & Scalable Systems

- 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

- Apache YARN
  - http://hadoop.apache.org/docs/current/
- Hadoop: The Definitive Guide
  - By Tom White
  - O’Reilly Publishing

Thank You!