Data-intensive computing systems

Hbase

MongoDB

University of Verona
Computer Science Department

Damiano Carra

Acknowledgements

- *Credits*
  - *Part of the course material is based on slides provided by the following authors*
    - *Mohamed Eltabakh, Susan B. Davidson, Kevin Matulef*

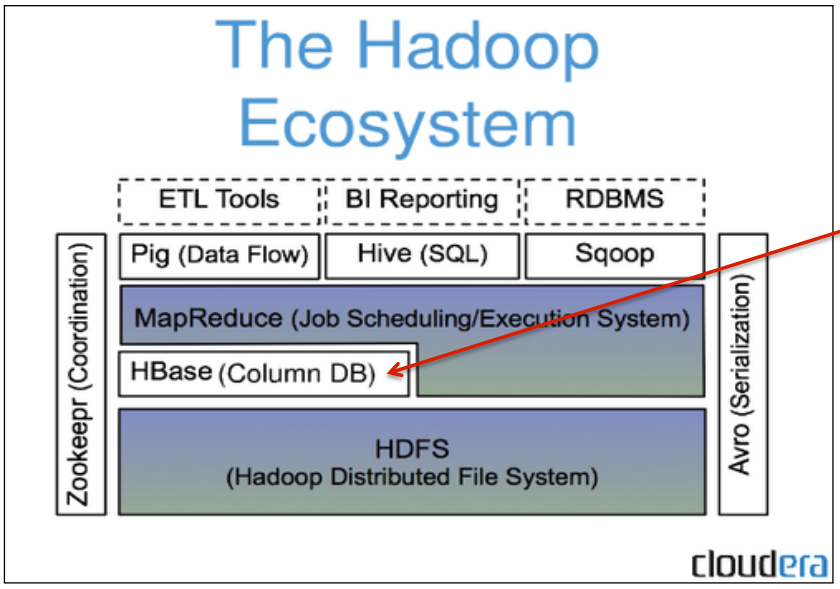
Introduction

- HDFS focuses on write once, read many workloads
- What if we need to store data and, in addition to full scans for analytics, occasionally update and read random elements?
- Example: Webtable
  - Output of the web crawler: for each web page URL, store the page, the attributes (language, MIME types), ...
  - The whole table is accessed by MapReduce jobs for analytics
  - Random rows are updated by the crawler
  - Random rows are read by the crawler or by other systems
- Solution: HBase

HBase: Overview

- HBase is a distributed column-oriented data store built on top of HDFS
- HBase is an Apache open source project whose goal is to provide storage for the Hadoop Distributed Computing
- Data is logically organized into tables, rows and columns

HBase: Part of Hadoop's Ecosystem



HBase is built on top of HDFS

HBase files are internally stored in HDFS

HBase vs. HDFS

- Both are distributed systems that scale to hundreds or thousands of nodes
- HDFS is good for batch processing (scans over big files)
  - Not good for record lookup
  - Not good for incremental addition of small batches
  - Not good for updates
- HBase is designed to efficiently address the above points
  - Fast record lookup
  - Support for record-level insertion
  - Support for updates (not in place)
- HBase updates are done by creating new versions of values

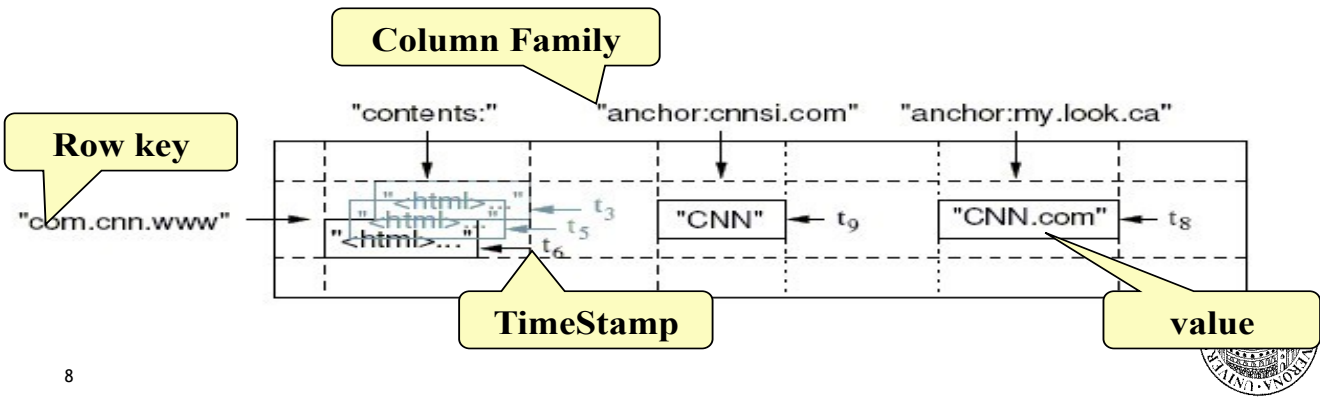
HBase vs. HDFS (Cont'd)

	Plain HDFS/MR	HBase
Write pattern	Append-only	Random write, bulk incremental
Read pattern	Full table scan, partition table scan	Random read, small range scan, or table scan
Hive (SQL) performance	Very good	4-5x slower
Structured storage	Do-it-yourself / TSV / SequenceFile / Avro / ?	Sparse column-family data model
Max data size	30+ PB	~1PB

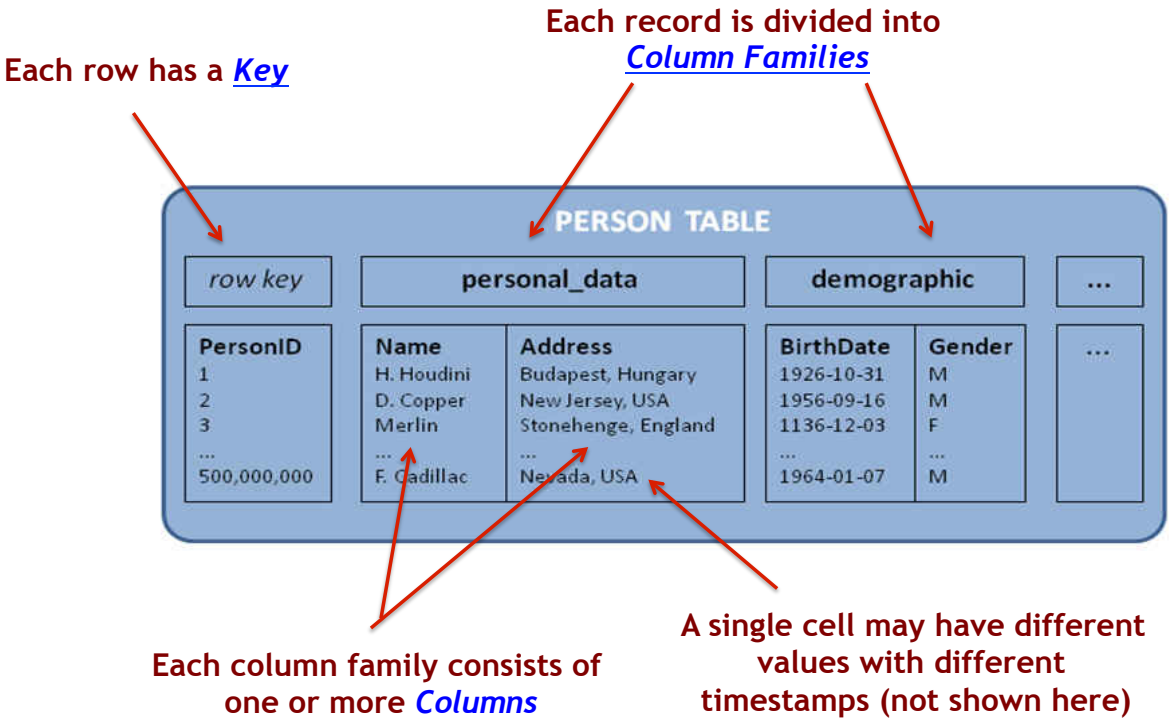
If application has neither random reads or writes ➔ Stick to HDFS

HBase Data Model

- HBase is based on Google's Bigtable model
  - Key-Value pairs
  - Rows are ordered by keys
- Table schema only define it's column families
  - Each family consists of any number of columns
  - Each column consists of any number of versions
  - Columns only exist when inserted,
  - Everything except table names are byte[]
  - (Row, Family: Column, Timestamp) → Value



HBase: Keys and Column Families



Notes on Data Model

- HBase schema consists of several Tables
- Each table consists of a set of Column Families
  - Once the Column Families are defined, Columns can be added at any time
    - HBase has Dynamic Columns
  - Hardcoded name convention
    - column_family_name:column_ID
- Not all the columns are used by all the keys
  - Table can be very sparse, many cells are empty

	cf1:c1	cf1:c2	cf2:c1	cf2:c2	cf2:c3
r1	■				
r2		■	■		
r3	■				■
r4	■	■		■	
r5		■		■	

HBase Physical Model

- Each column family is stored in a separate file (called HTables)
- Key & Version numbers are replicated with each column family
- Empty cells are not stored

HBase maintains a multi-level index on values: *<key, column family, column name, timestamp>*

Table 5.3. ColumnFamily `contents`

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

Table 5.2. 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"

Example

Row key	Data
cutting	info: { 'height': '9ft', 'state': 'CA' } roles: { 'ASF': 'Director', 'Hadoop': 'Founder' }
tlipcon	info: { 'height': '5ft7, 'state': 'CA' } roles: { 'Hadoop': 'Committer'@ts=2010, 'Hadoop': 'PMC'@ts=2011, 'Hive': 'Contributor' }

`info` **Column Family**

Row key	Column key	Timestamp	Cell value
cutting	info:height	1273516197868	9ft
cutting	info:state	1043871824184	CA
tlipcon	info:height	1273878447049	5ft7
tlipcon	info:state	1273616297446	CA

`roles` **Column Family**

Row key	Column key	Timestamp	Cell value
cutting	roles:ASF	1273871823022	Director
cutting	roles:Hadoop	1183746289103	Founder
tlipcon	roles:Hadoop	1300062064923	PMC
tlipcon	roles:Hadoop	1293388212294	Committer
tlipcon	roles:Hive	1273616297446	Contributor

Sorted on disk by Row key, Col key, descending timestamp

Milliseconds since unix epoch

cloudera

Column Families

- Different sets of columns may have different properties and access patterns

- Configurable by column family:
  - Compression (none, gzip, …)
  - Version retention policies

- Column Families stored separately on disk
  - Access one without wasting I/O on the other

HBase Regions

- Each HTable (column family) is partitioned horizontally into regions
  - Regions are counterpart to HDFS blocks

Table 5.3. ColumnFamily `contents`

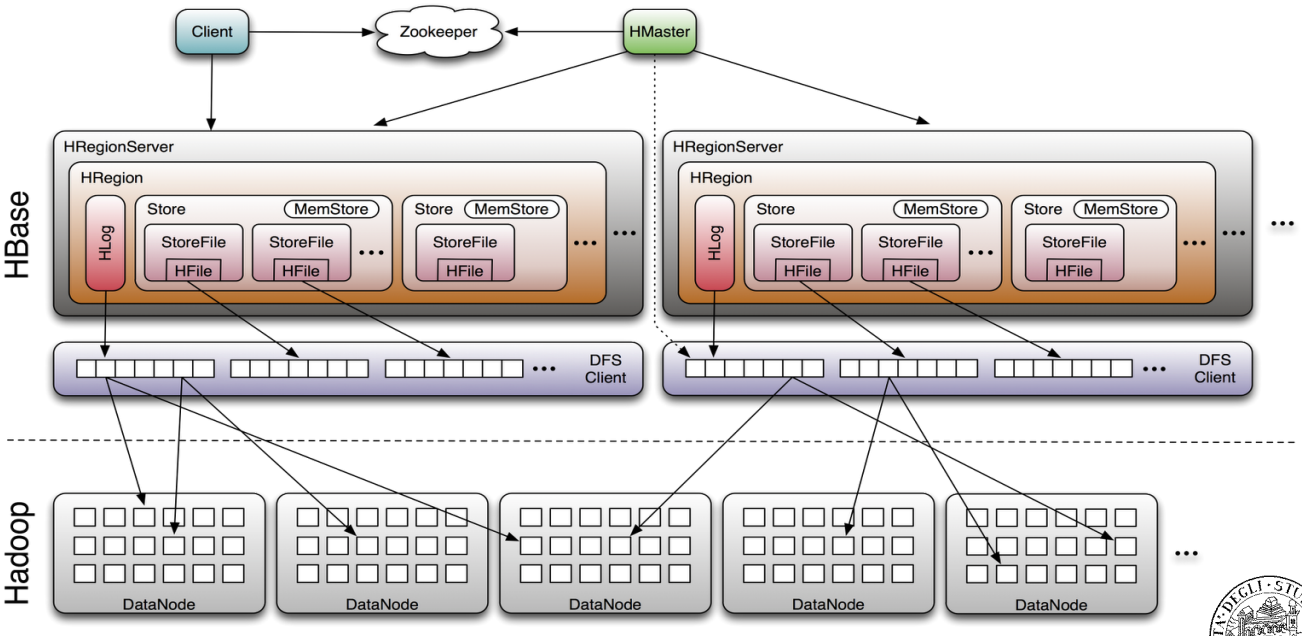
Row Key	Time Stamp	ColumnFamily "contents:"
"com.cnn.www"	t6	`contents:html` = "<html>..."
"com.cnn.www"	t5	`contents:html` = "<html>..."
"com.cnn.www"	t3	`contents:html` = "<html>..."

Each will be one region

HBase Components

- Region
  - A subset of a table's rows, like horizontal range partitioning
  - Automatically done
- RegionServer (many slaves)
  - Manages data regions
  - Serves data for reads and writes (using a log)
- Master
  - Responsible for coordinating the slaves
  - Assigns regions, detects failures
  - Admin functions

Big Picture



Access to data

- Set of APIs that can be used to do basic operations
- Put()
  - Insert a new record (with a new key), or insert a record for an existing key
  - The column ID (given an existing column family) must be also specified
  - Implicit or explicit versioning
- Get()
  - retrieve the value given a key and a column ID
- Scan()
  - retrieve all the values given a column identifier (and a range of keys)
- Delete()
  - Multiple levels
    - Can mark an entire column family as deleted
    - Can make all column families of a given row as deleted

Access to data: Get()

Select value from table where key= 'com.apache.www' AND label= 'anchor:apache.com'

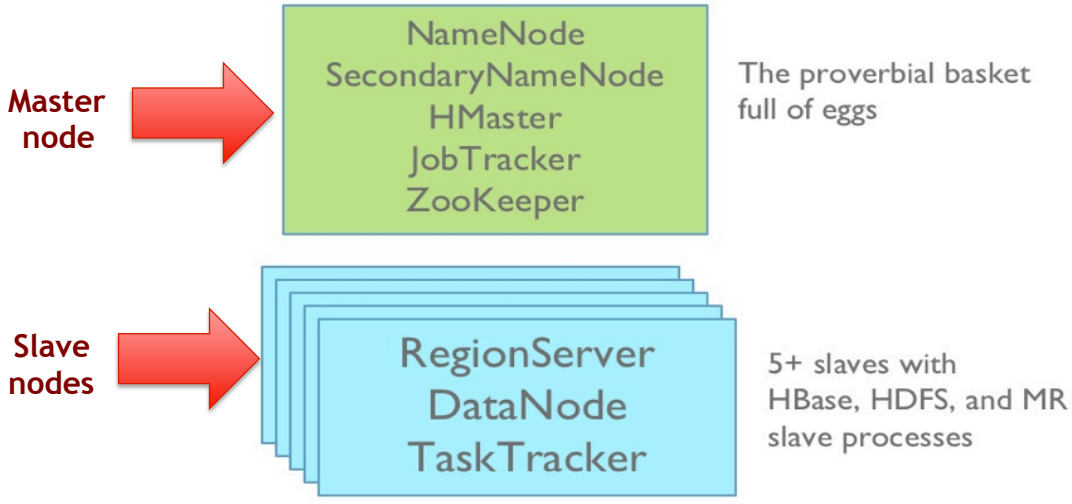
Row key	Time Stamp	Column "anchor:"	
"com.apache.www"	t12		
	t11		
	t10	"anchor:apache.com"	"APACHE"
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"
	t6		
	t5		
	t3		

Access to data: Scan()

Select value from table where anchor= 'cnnsi.com'

Row key	Time Stamp	Column "anchor:"	
"com.apache.www"	t12		
	t11		
	t10	"anchor:apache.com"	"APACHE"
"com.cnn.www"	t9	"anchor:cnnsi.com"	"CNN"
	t8	"anchor:my.look.ca"	"CNN.com"
	t6		
	t5		
	t3		

HBase Deployment



HBase vs. RDBMS

	RDBMS	HBase
Data layout	Row-oriented	Column-family-oriented
Transactions	Multi-row ACID	Single row only
Query language	SQL	get/put/scan/etc *
Security	Authentication/Authorization	Work in progress
Indexes	On arbitrary columns	Row-key only
Max data size	TBs	~1PB
Read/write throughput limits	1000s queries/second	Millions of queries/second

MongoDB

Overview

- Document oriented, not table/row oriented
- Collection of binary JSON (BSON) documents
- Schemaless
- No relations or transactions native in database
- Scalable and high-performance
- Full index support
- Written in C++
- Servers for all major platforms
- Drivers for all major development environments
- Free and open-source, but also commercial support

BSON

- Binary JSON
- Binary encoded serialization of JSON-like documents
- Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays. BSON also contains extensions that allow representation of data types that are not part of the JSON spec. For example, BSON has a Date type and a BinData type.
- The driver performs translation from the language's "domain object" data representation to BSON, and back

MondoDB Data Model and Types

- Data Model
  - A MongoDB deployment hosts a number of databases
  - A database holds a set of collections
  - A collection holds a set of documents
  - A document is a *set* of key-value pairs

- Data types
  - Basic: Null, Boolean, Integer (32- and 64-bit), Floating point, String
  - More complex: Date, Code (JavaScript), Array, Embedded document

Sample Document

Always indexed. Can be any BSON data type other than an array

```
{
   _id: ObjectId("4c4ba5c0672c685e5e8aabf3'),
   author: "Kevin",
   date: new Date("02/28/2012"),
   text: "About MongoDB…",
   tags: [ "tech", "databases"]
   comments: [{author: "jim", comment: "I disagree" },
              {author: "nancy", comment: "Good post"}
             ]
}
```

Array of documents

Core MongoDB Operations

- CRUD: create, read, update, and delete
- Insert
  - One at a time: `db.mycollection.insert(mydoc)`
  - Batch insert
- Querying
  - Use `find()/findOne()` functions and a query document
  - Ranges, set inclusion, inequalities using $ conditionals
- Delete
  - Documents that match some predicate, e.g. to remove the document just added

    ```
    db.mycollection.remove({"_id": 1})
    ```
  - All documents in a collection

    ```
    db.mycollection.remove()
    ```


Find

- Get the entire collection (called posts)

  ```
  db.posts.find()
  ```

- Get the documents satisfying some constraints

  ```
  db.posts.find({"author": "Kevin"})
  ```

- Specifying Which Keys to Return

  ```
  db.mydoc.find({}, {"author", "tags"})
  ```

  ```
  {
     _id: ObjectId("4c4ba5c0672c685e5e8aabf3"),
     author: "Kevin",
     tags: [ "tech", "databases"]
  }
  ```


Ranges, Negation, OR-clauses

- Comparison operators: $lt, $lte, $gt, $gte

```
start = new Date(01/01/2012)
end = new Date(04/01/2012)
db.posts.find({“date”: {“$gte”: start, “$lte”: end}})
```

- Negation: $ne

```
db.posts.find({“date”: {“$ne”: end}})
```

- Or queries: $in (single key), $or (different keys)

```
db.posts.find({“date”: {“$in”: [start, end]}})
db.posts.find({“$or”:[{“date”: start}, {“name”: “John”}]})
```

Limits, Skips, Sort, Count

- Limits the number of results to 3

```
db.posts.find().limit(3)
```

- Skips the first three results and returns the rest

```
db.posts.find().skip(3)
```

- Sorts by author ascending (1) and title descending (-1)

```
db.posts.find().sort({“author”:1, “title”: -1})
```

- Counts the number of documents in the people collection matching the find(…)

```
db.people.find(…).count()
```

Summary of MongoDB

- MongoDB is an example of a document-oriented NoSQL solution
- The query language is limited, and oriented around "collection" (relation) at a time processing
  - Joins are done via a query language
- The power of the solution lies in the distributed, parallel nature of query processing
  - Replication and sharding