Alternative to relational database system - NoSQL

Not Only SQL

Bernard Lee Kok Bang

Non-relational databases – Not only SQL

- Run into some limitations with SQL and relational database systems [meant for analytic query, such as business report at the end of the month]

- Don't need the ability to issue arbitrary queries across entire dataset

- Need the ability to very quickly answer a specific question such as "What movie should I recommend for this customer", or "What web pages has this customer looked at in the past"...

- Need an answer at a very large scale very quickly across a massive dataset [handle tens of thousands of transactions per second]

- Build to scale horizontally, very fast, and very resilient

-Analytical Query
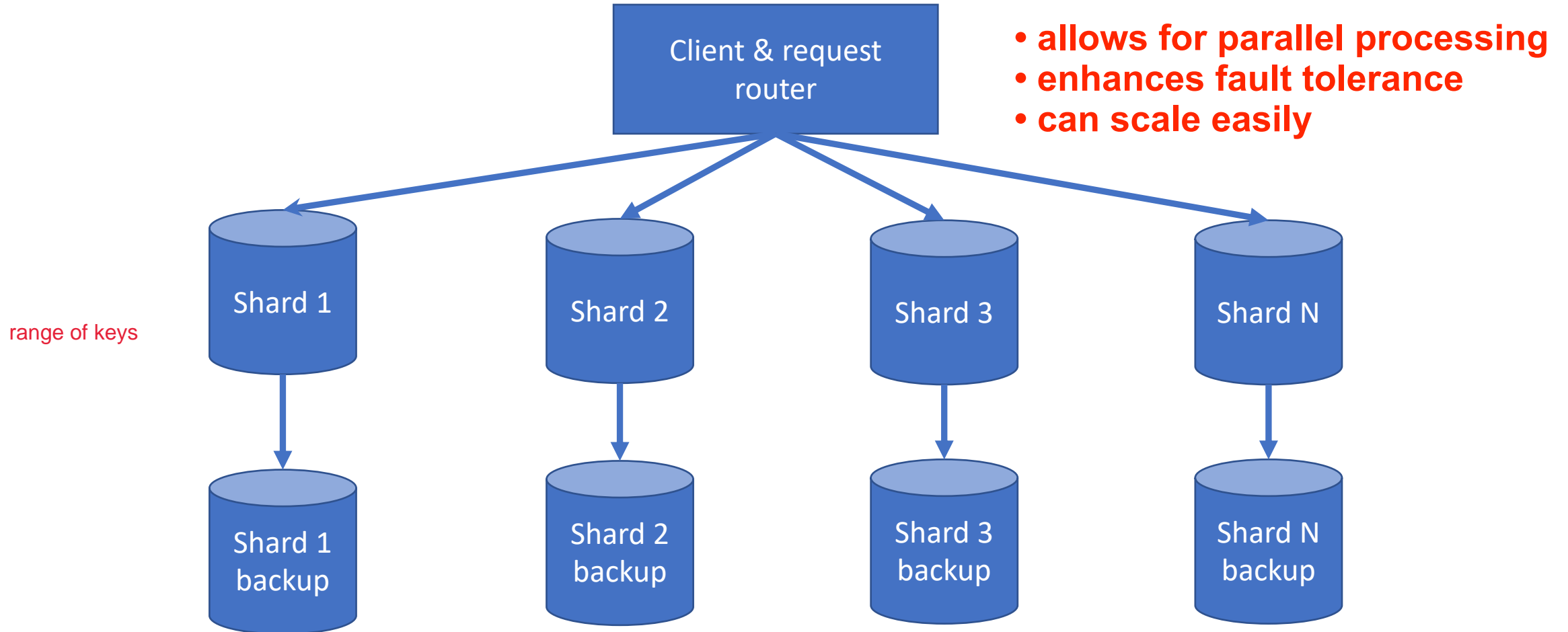-Transactional query (very fast)

Scaling up MySQL etc. to massive loads requires extreme measures

-put everything in one table
- Denormalization [increase efficiency]

-talk about spark
- Caching layers [distributed in-memory caching layer]

-distributed system
- Master/slave setups [require DBA to maintain]

-separate base on certain key
- Sharding [split database into range partition, specific database handles specific ranges of keys -> manual process]

- Materialized views [anticipate data in the expected format]

Do we really need SQL?

- High-transaction queries are probably pretty simple once de-normalized [such as using JSON file format]

- A simple get/put API may meet our needs [*"given this customer ID, give me back this list of information"*]

- Looking up values for a given key is simple, fast, and scalable [at high level]

- Questions:
  - *What's the item information associated with this item?*
  - *What pages have this customer looked at?*

- Don't need a rich query language for answering the above question

- Don't need big fancy relational database; instead, we need more scalable system that can easily horizontally partitioned for given key ranges
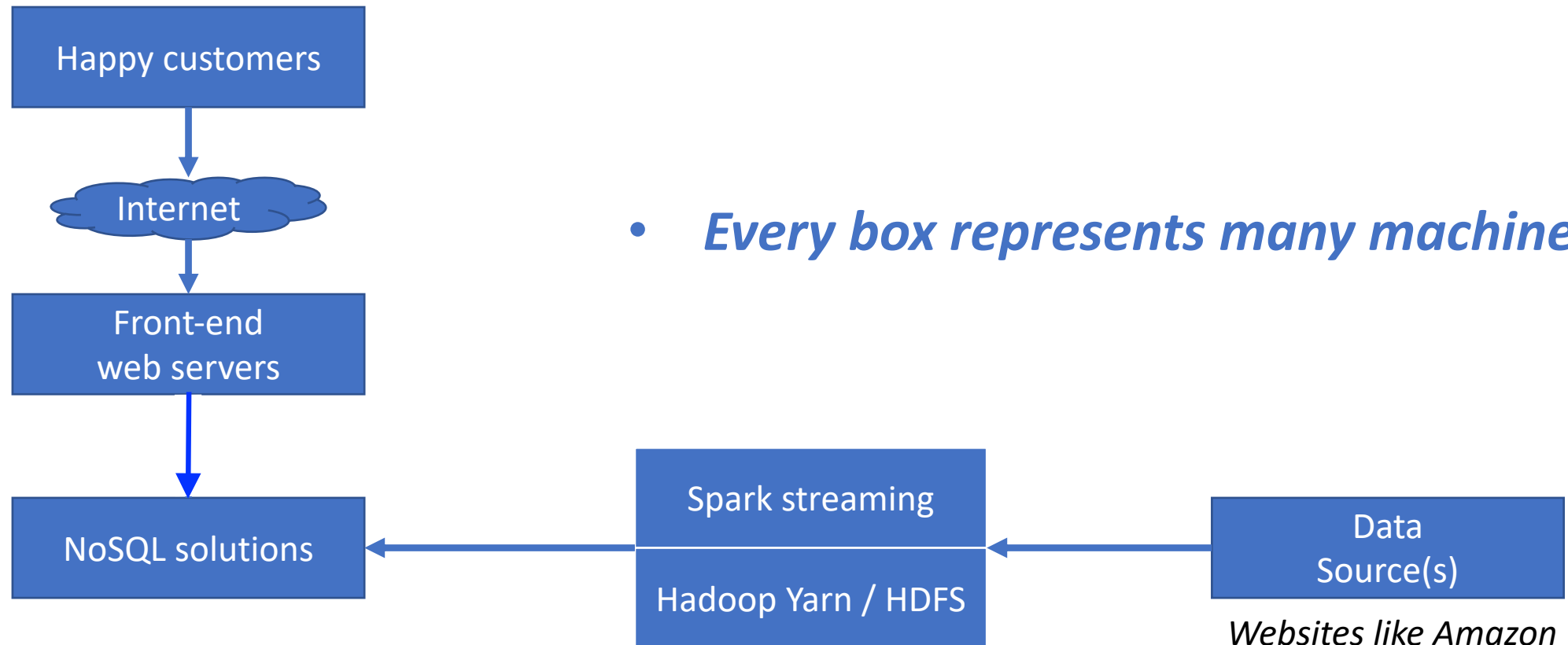
NoSQL database sample architecture

Client & request router

- **allows for parallel processing**
- **enhances fault tolerance**
- **can scale easily**

range of keys

Shard 1

Shard 2

Shard 3

Shard N

Shard 1 backup

Shard 2 backup

Shard 3 backup

Shard N backup

5

Use the right tool for the job

- For *analytic queries*, Hive, Pig, Spark, etc. work great [business report]

- Exporting data to MySQL is plenty fast for most applications too

- But if we work at *giant scale* [Amazon, Google…] - export our data to a non-relational database for fast and scalable serving of that data to web applications, etc.

Sample application architecture (simplified)

- **Every box represents many machines**

Happy customers

Internet

Front-end web servers

NoSQL solutions

Spark streaming

Hadoop Yarn / HDFS

Data Source(s)

Websites like Amazon

HBase

- Non-relational, scalable database built on HDFS

- Expose massive data sitting on HDFS to web service or web application [that can operate very quickly and high scale]

- Don't have a query language

- Have an API that can quickly answer the question
  Programming Interface
  - "what are the values for this key"
  - "store this value for this key"

Based on Google's BigTable (2008)

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google called Bigtable. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data.

In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

2 Data Model

A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.
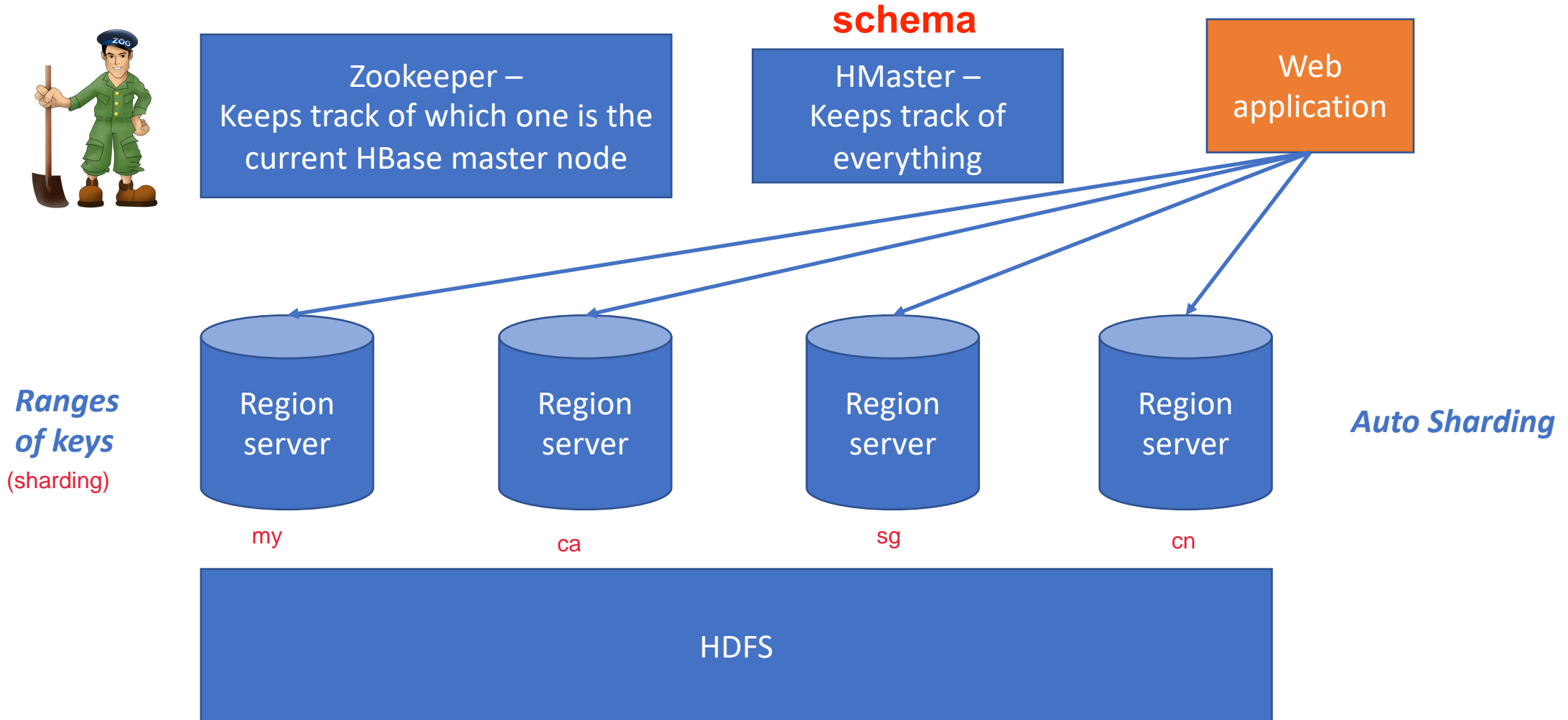
(row:string, column:string, time:int64) → string

- Problem of storing information about links to all the web pages on the entire planet -> big data problem
- HBase implements the application of BigTable
  - *Open-source implementation of the algorithms and systems*

Chang, Fay, et al. (2008)

CRUD operations

- Create

- Read

- Update

- Delete

- No query language, only CRUD API

- Offers transactional access -> *FAST & LARGE SCALE*!!!

HBase architecture

schema

Zookeeper –
Keeps track of which one is the
current HBase master node

HMaster –
Keeps track of
everything

Web application

*Ranges
of keys*

(sharding)

Region server

Region server

Region server

Region server

Auto Sharding

my

ca

sg

cn

HDFS

HBase data model

- Fast access to any given *ROW*

- A ROW is referenced by a unique *KEY*

- Each ROW has some small number of *COLUMN FAMILIES*

- A COLUMN FAMILY may contain arbitrary *COLUMNS [can deal with sparse data]*

- Can have a very large number of COLUMNS in a COLUMN FAMILY

- Each *CELL* can have many *VERSIONS* with given timestamps *[can store history]*

- Sparse data is OK – missing columns in a row consume no storage

In reverse order:
- accessing the most recent or latest data first
- interested in the most recent entries in the database
- simplifies and speeds up the retrieval process

arrange salary payslip

Key [reverse order]

com.cnn.www

- *Track all the links that connect to a given web page*

Contents column family

Contents:

<html><head>
<html><head>
<html><head>
CNN...

} 3 versions
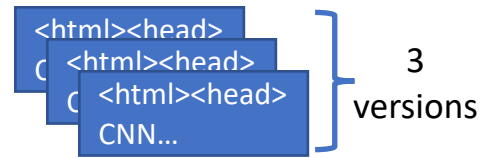
Anchor column family

Anchor: cnnsi.com

"CNN"

Anchor: my.look.ca

"CNN.com"

Syntax → column family:column name

Anchor: a piece of text which marks the beginning and/or the end of a hypertext link.

12

Using HBase Shell - PuTTY

#Opens the HBase shell
hbase shell

#Lists down all the tables present in HBase
list

#Creates a new table
create 'newtbl', 'knowledge'

#Checks if the table was created
describe 'newtbl'

#Checks the status of HBase
status 'summary'

#Put some data into the 'newtbl'
put 'newtbl', 'r1', 'knowledge:sports', 'cricket'
put 'newtbl', 'r1', 'knowledge:science', 'chemistry'
put 'newtbl', 'r1', 'knowledge:science', 'physics'
put 'newtbl', 'r2', 'knowledge:economics', 'macro economics'
put 'newtbl', 'r2', 'knowledge:music', 'pop music'

Using HBase Shell – PuTTY (cont...)

#List the contents of the 'newtbl'
scan 'newtbl'

#Checks if the table is enabled
is_enabled 'newtbl'

#Disables the table
disable 'newtbl'

#Lists the contents of the 'newtbl'. Note that this will throw an error as the table is disabled.
scan 'newtbl'

#Updates column family in the table
alter 'newtbl', 'test_info'

#Enables the table
enable 'newtbl'

#Checks the column families after updating
describe 'newtbl'

#Extracts the values for r1 in the 'newtbl'
get 'newtbl', 'r1'

Using HBase Shell – PuTTY (cont...)

#Adds new information to r1 for economics. Note that this will update the table but will not override the information
put 'newtbl', 'r1', 'knowledge:economics', 'market economics'

#Displays the results for r1
get 'newtbl', 'r1'

#Count number of rows in 'newtbl'
count 'newtbl'

 # Delete the 'newtbl'. Note: Error because the 'newtbl' is still enabled
drop 'newtbl'

#Disable first, then only delete the the 'newtbl'
disable 'newtbl'

 # Check if the 'newtbl' still available
list

What if I wanted to add in more values in same row?

```
# Disable the table if it exists
disable 'newtbl'

# Alter the table to keep up to 3 versions of each cell
alter 'newtbl', NAME => 'knowledge', VERSIONS => 3

# Enable the table
enable 'newtbl'

# Put the values
put 'newtbl', 'r1', 'knowledge:sports', 'cricket'
put 'newtbl', 'r1', 'knowledge:sports', 'football'
put 'newtbl', 'r1', 'knowledge:sports', 'basketball'

# Retrieve the versions
get 'newtbl', 'r1', {COLUMN => 'knowledge:sports', VERSIONS => 3}
```

Using HBase Shell – PuTTY (cont...)

#Check current HBase user
whoami

#Different HBase table command usages and its syntaxes
table_help

#HBase status
status

#HBase version
version

Let's play with HBase

- Create a HBase table for *movie ratings grouped by user*
- Show how we can quickly query for individual users
- Good example of sparse data

Colum family:rating

UserID	Rating:50	Rating:33	Rating: 223
	1	5	5
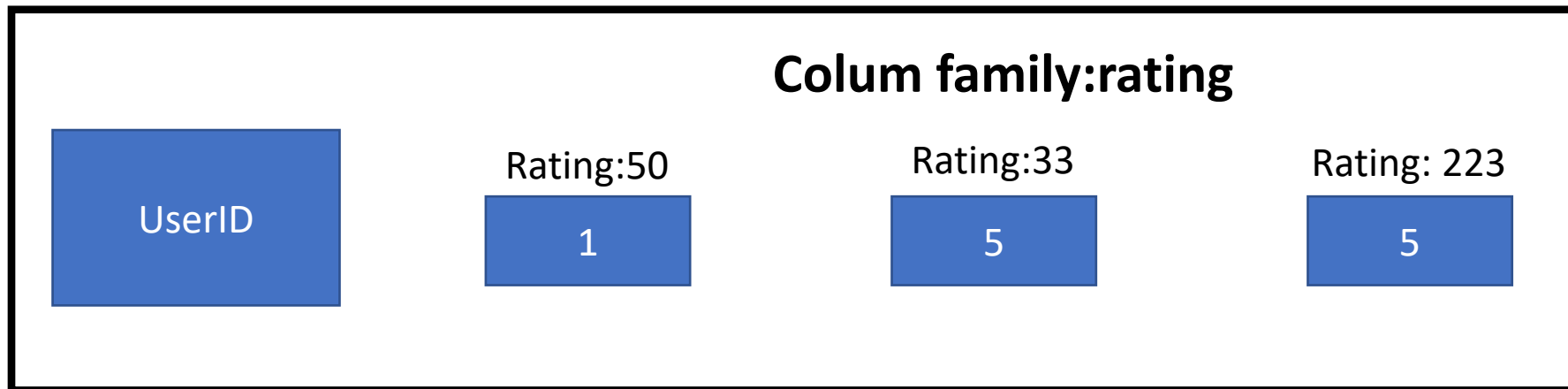
How are we doing it?

REST API

What is Rest API?

DATABASE → WEB SERVER → RESTFUL API → YOUR WEBSITE APPLICATION
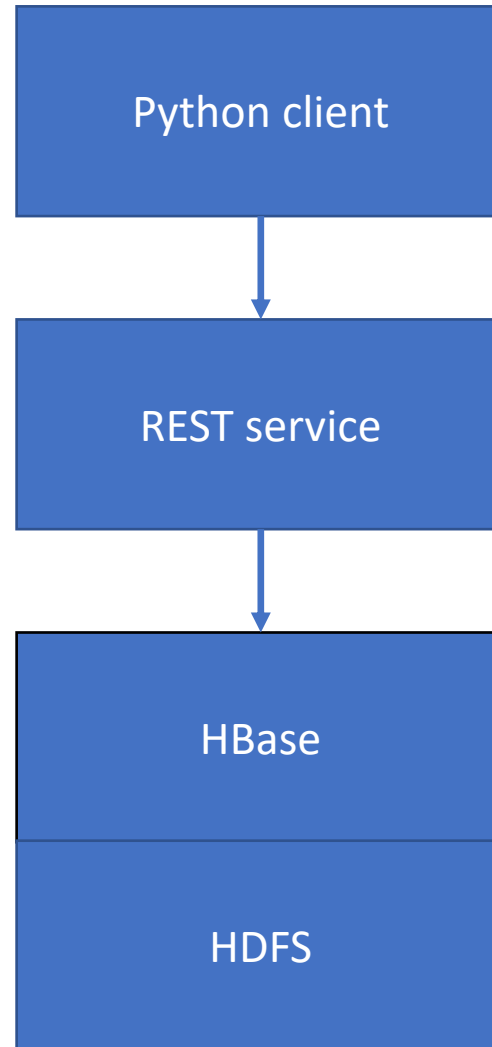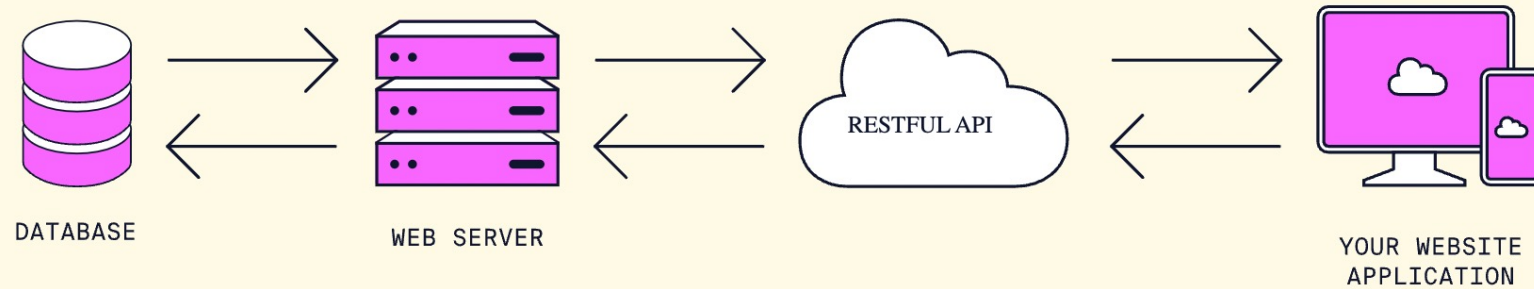
- providing standards between computer systems on the web, making it easier for systems to communicate with each other

Change some settings: Port 8000 & Ambari

- Open a port so that Python script can talk to REST service

1. Right click Horthonworks Docker Sandbox on VirtualBox

2. Choose Settings, Network, Advanced, Port Forwarding

3. Click the ADD button and add in the following details

   →Name: *HBase REST*; Protocol: *TCP*; Host IP: *127.0.0.1*; Host Port:*8000*; Guest Port: *8000*

4. If Port 8000 has been occupied, then just change the Name to *HBase REST*

5. Log in to Ambari using *admin* username

6. Choose *HBase* [left panel], and *start* the HBase under *Service Action*

Change some settings: PuTTY

Starts REST server sitting on top of HBase and HDFS

1. Login as maria_dev

   su root
   Password:
   /usr/hdp/current/hbase-master/bin/hbase-daemon.sh start rest -p 8000 --infoport 8001

Start coding

```
pip install starbase

from starbase import Connection

c = Connection("127.0.0.1", "8000")

ratings = c.table('ratings')

if (ratings.exists()):
        print ("Dropping existing ratings table\n")
        ratings.drop()

ratings.create('rating')

print("Parsing the ml-100k ratings data... \n")
ratingFile = open(r"c:/Downloads/ml-100k/ml-100k/u.data", "r")

batch = ratings.batch()
```

Change directory accordingly

```
for line in ratingFile:
        (userID, movieID, rating, timestamp) = line.split()
        batch.update(userID, {'rating': {movieID: rating}})

ratingFile.close()

print("Committing ratings data to HBase via REST service\n")
batch.commit(finalize=TRUE)
                        True

print("Get back ratings for some users...\n")
print("Ratings for user ID 33:\n")
print(ratings.fetch("33"))

ratings.drop()
```
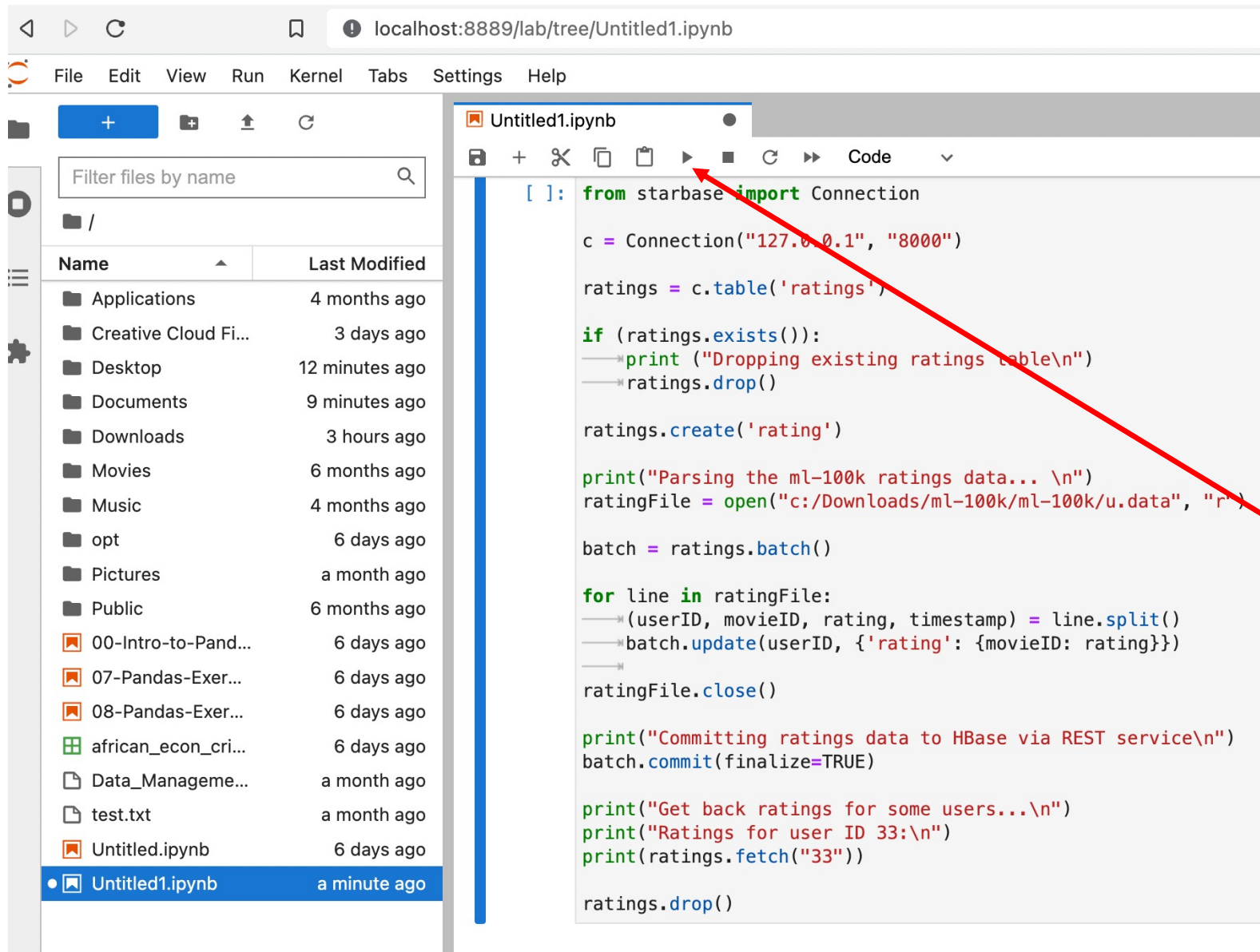
Using JupyterLab

Click the icon to start running the python script

26

Output

'185': '4', '188': '3', '189': '3', '4': '3', '97': '3', '6': '5', '94': '2', '99': '3',
'228': '5', '227': '4', '165': '5', '166': '5', '224': '5', '223': '5', '222': '4', '221'
'5', '12': '5', '15': '5', '14': '5', '17': '3', '16': '5', '19': '5', '18': '4', '272':
'5', '153': '3', '152': '5', '155': '2', '154': '5', '157': '4', '156': '4', '159': '3',
'239': '4', '83': '3', '234': '4', '235': '5', '236': '4', '237': '2', '230': '4', '231':
'3', '46': '4', '47': '4', '44': '5', '45': '5', '42': '5', '43': '4', '40': '3', '118':
'146': '4', '200': '3', '203': '4', '202': '5', '205': '3', '204': '5', '207': '5', '206'
'149': '2', '77': '4', '76': '4', '75': '4', '74': '1', '73': '3', '72': '4', '71': '3',
'2': '3', '263': '1', '262': '3', '261': '1', '41': '2', '260': '1', '267': '4', '67': '3
Ratings for user ID 33:

{'rating': {'751': '4', '880': '3', '339': '3', '895': '3', '313': '5', '872': '3', '333'
'258': '4', '260': '4', '678': '4', '328': '4', '288': '4', '292': '4', '879': '3', '300'
'329': '4', '348': '4', '682': '4'}}

In [2]:

Ratings for user ID 33

To stop REST interface:
/usr/hdp/current/hbase-master/bin/hbase-daemon.sh stop rest