Columbia University

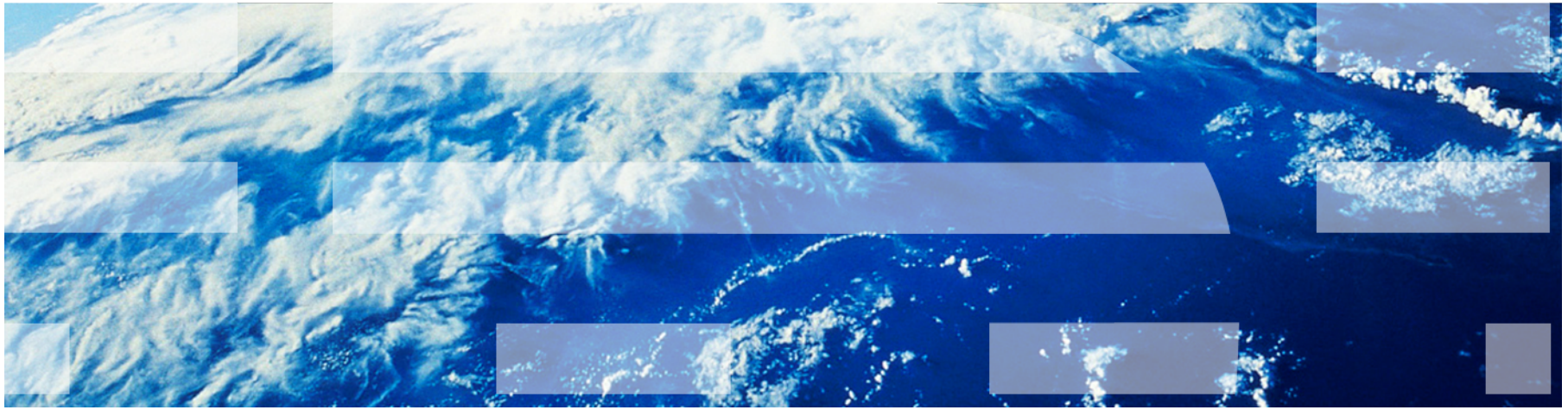
E6893 Big Data Analytics Lecture 3:

Big Data Storage and Processing

Ching-Yung Lin, Ph.D.

Adjunct Professor, Dept. of Electrical Engineering and Computer Science

Distinguished Research and Chief Scientist, Graph Computing, IBM Research



September 24th, 2015

TA Office Hours:

Location:

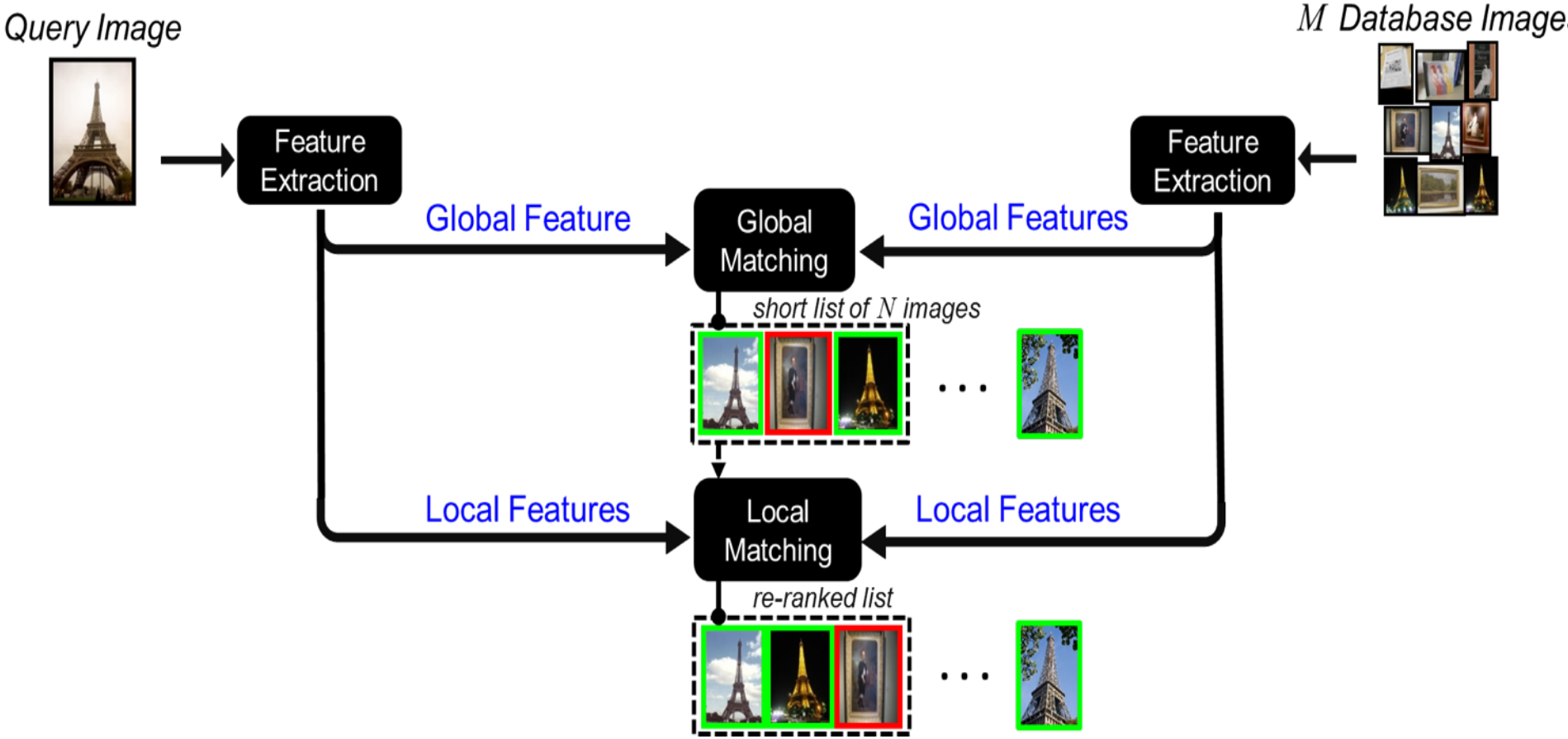
Computer Science TA room: Mudd 122A

Monday 9-11am: Ghazal Fazelnia
Monday 3:30-5:30pm: Rama Kompella
Tuesday 9:30-11:30am: Siyuan Zhang
Tuesday 4-6pm: Tian Han
Wednesday 4-6pm: Yongchen Jiang
Thursday 9:30-11:30am: Liqun Chen
Friday 9:30-11:30am: Rishina Tah
Friday 4-6pm: Junkai Yan

Potential Project Directions

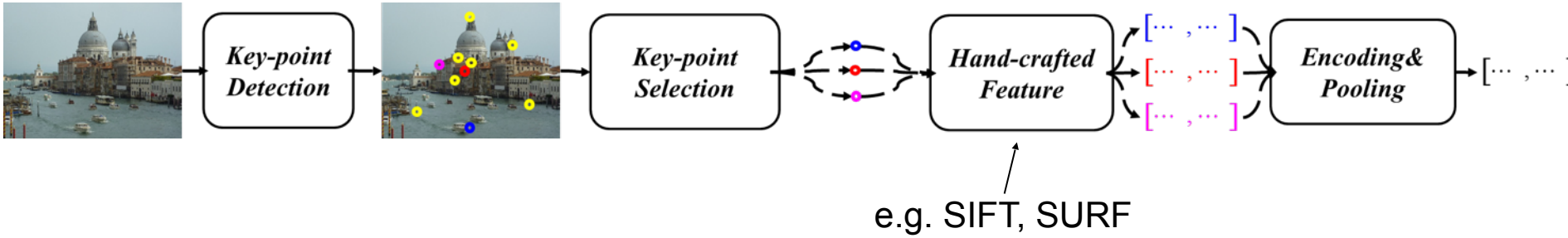
- Contact: Wen-Hsiao (Wen) Peng, <wpeng@us.ibm.com>
- Projects:
  - Project 1: Large-scale Visual Search
    - Retrieving similar images from a database using a query image as input is a classical problem in computer vision. The task involves image content matching, and can be challenging because the query image and its corresponding references in the database may exhibit differences in view angle, lighting conditions, spatial resolution, representation fidelity, etc. Moreover, the rising of large-scale image databases calls for low-complexity matching algorithms. Over the years, a significant amount of work has been proposed to address these issues. Most extract low-level features, such as SIFT, SURF, etc, from an image, convert them into a compact image representation for fast search, and perform graph matching with respect to their spatial locations for re-ranking the retrieved images. Low-level features, however, may give poor retrieval results due to their deficiencies to capture high-level semantic information. Motivated by the great success of Convolutional Neural Networks (CNN) in detecting mid-level objects/ parts in images, this project aims to extend its use to visual search for video applications.
    - Basic knowledge of image/video processing and computer vision

Image Retrieval Pipeline: An Example

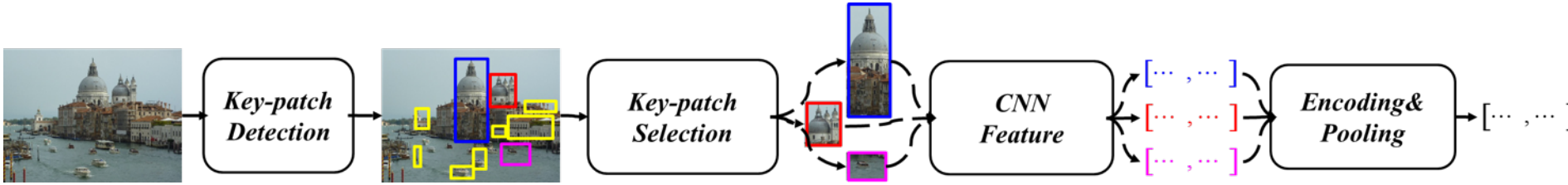


Focus of This Project: Global Feature and CNN

How it is done currently (e.g. MPEG CDVS)



What you may consider …



More you could think about -- video search

Potential Project Directions

- Contact: Yinglong Xia, <yxia@us.ibm.com>
- Projects:
  - Project 1: Goozie – a DAG workflow scheduler for IBM System G gShell plugins
    - Description
      Create a DAG for workflow management and scheduling, where each vertex represent a task (a plugin in IBM System G gShell) and the vertex property provides the task description; the edges represent precedency constraints. Scheduling the tasks according to the DAG completes an analytics consisting of a set of gShell commands and the scheduling shall invoke tasks concurrently if no precedency constraint. Tasks can be of different gShell plugin types, such as those for GPU/
    - Prior experience required
      Basic knowledge on Apache Oozie; Skillful C++/C++11 coding experience; Decent knowledge on IBM System G gShell

- Project 2: BlinkGraph – A graph storage framework supporting queries with bounded error rate or response time
  - Description
    Perform sampling on IBM System G native store and store the graph data hieratically to support time bound in graph queries; the sampling results are stored onto disk and can be loaded into memory if necessary, according to the response time bound or the error bound specified by users. Lower error rate bound leads to higher query time; equivalently, lower response time bound results in higher error rate bound. This allows users to specify different bound in practice to meet the requirement. For example, the user may want a plugin return result within 2 seconds, even if the more accurate result has not achieved yet.
  - Prior experience required
    Basic knowledge on BlinkDB; Familiar with Graph sampling techniques; C++ coding and debugging skills; basic knowledge on sampling.

- Project 3: PanNoSQL – Unify typical NoSQL DBs using an advanced property management in GraphDB
  - Description
    Extend IBM System G native store, a property graph store, to develop an advanced property management that unifies typical NoSQL DBs, including graph store, Key-Value store, Column store, and Document store. The basic idea is to bring rich property management functionality, so that the property is more than a set of key-value pairs, but can be of a set of key-value pairs, even a JSON document. The enhanced expressiveness brings a real NoSQL store.
  - Prior experience required
    Basic knowledge on Key-value store (BerkeleyDB, LMDB, etc.); Familiar with GraphDB (neo4j, IBM System G, etc.); skillful C++ coding and debugging capability; basic knowledge on performance tuning.

Potential Projects

Contact: Toyo Suzumura, tsuzumura@us.ibm.com

Projects:

Project 1: Scalable Graph Computing Platform with GraphX/Spark and IBM System G

The project aims at designing and implementing a scalable graph computing platform based upon GraphX/Spark and IBM System G. The platform includes a new programming model such as incremental graph processing models and fine-grained graph query processing modesl, as well as novel performance optimization ideas such as GPU-accelerated graph processing, I/O optimization, differential graph processing, an efficient and tightly coupled computation with graph databases such as IBM System G Native Store

Prior experience required: C/C++, Java / Java Performance Profiler, Distributed Systems, Linux

Potential Project Directions

- Contact: Sabrina Lin / sabrinal@us.ibm.com
- Projects:
  - Project 1: Cross-source social media event detection
    - Identify events in Twitter based on sentiment and the statistics of the tweets, then validate the events with local news. Potential extension of this project could include a summarization and visualization of the two data sources and the detected events.
    - Machine learning
  - Project 2: Tamper-invariant image feature extraction via deep networks
    - Deep learning has shown success in extracting high-level image features for various computer vision tasks. However these features are not robust against rotation, clipping, or other image tempering methods. This project aims to study the deep network based image features and design the network to extract tamper-invariant features.
    - Machine learning

Potential Project Directions

- Contact: Conglei Shi / conglei.shi@us.ibm.com
- Projects:
  - Project 1: A Visual Analytics System of the deep learning process
    - Description: Deeping learning has shown its great power in a lot of areas such as speech and vision. So far, few research has been done on deeply understanding the learning progress, as well as how the structure of networks and parameters affect the performance. In this project, we try to build a visual analytics framework to help analysts to better understand the "black box" procedure.
  - Project 2: Visualization of spatial-temporal patterns of user tweeting behavior on information diffusion process
    - Description: Huge amount of twitter data now is available for analyst to understand user behaviors on online activity. In this project, we want to build a visual analytics system combining with data mining techniques to analyze the spatial-temporal information on diffusion process for specific event. Taking the presidential election for example, we can use this system to analyze how the interaction network according to their geo information builds over time and how the sentiment changes over time.

Potential Project Directions

- Contact: Larry Lai, larrylai@us.ibm.com
- Projects:
  - Project 1: Image recognition with a huge dataset on iOS devices
    - Description:
      Most people store thousands of pictures on their mobile devices. There is a scenario: a user takes one picture and would like to know if it is similar to any of the pictures in his/her device. So, the problem is how do you immediately analyzes the image similarity from a huge dataset on an iOS device. You may need to leverage OpenCV to calculate image features and build a table, like the CoreData on iOS, to reduce the computation. In this project, you are required to run your code on iOS devices.
    - Prior experience required:
      OpenCV and **iOS developing**

Potential Project Directions

- Contact: Eric Johnson, efjohnson@us.ibm.com
- Projects:
- Traversing Graph Structures to Predict 2+ Hop Social Connections (Twitter)

- Predicting Stock Breakouts Through Based on Historical Trends (PySpark / Google Finance)

- Gas Station Optimization App (Approximates gas savings (price) based on distance/price savings)

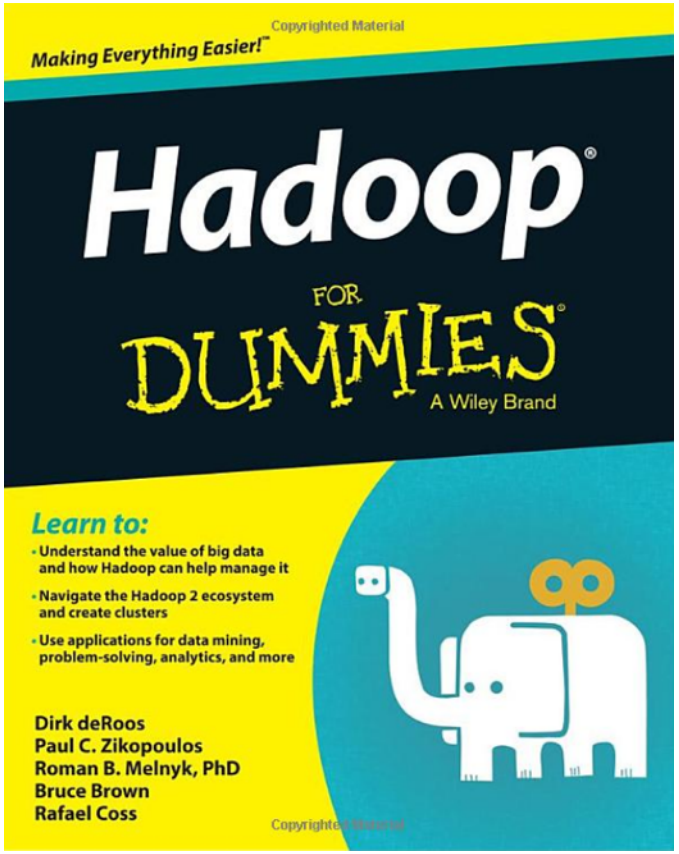
Potential Project Directions (Visual Analysis)

- Contact: Nan Cao nancao@us.ibm.com
- Projects:
  - Project 1: Visual Analysis of User Behaviors in Social Media
    - **Description**: In this project, we will design novel visualizations to visually summarize and compare different behaviors in Social Media such as Twitter, FaceBook and many other blogs. Example topics in this project including:
      Detecting sentiment divergence among users:
      http://nancao.org/pubs/lu_infovis13_poster_paper.pdf
      Finding anomalous user behaviors:
      http://nancao.org/pubs/cao_vast15_paper.pdf
      Summarizing user behaviors:
      http://nancao.org/pubs/cao_cga15_paper.pdf
    - **Prior experience required:** (1) familiar with front-end development techniques including javascript, css, html; (2) has fundamental knowledge about data visualization and has coding experiences withUI or visualization packages such as d3.js, jquery-ui, or angular.js. (2) has passing knowledge about data mining and machine learning as well as natural language processing; (3) be good at using data analysis packages such as scikit-learn.

Potential Project Directions (Visual Analysis)

- Project 2: Visual Analysis of Scholar data
  - **Description:** In this project we will finding and visualize nowadays' research trend based on scholar datasets collected from DBLP. Specific topic including detecting and visualize research communities, finding interdisciplinary research areas, as well as predict emerging research topics. One example project in this topic can be found here: http://nancao.org/pubs/cao_tvcg15_paper.pdf
  - **Prior experience required:** (1) familiar with front-end development techinques including javascript, css, html; (2) has fundamental knowledge about data visualization and has coding experiences withUI or visualization packages such as d3.js, jquery-ui, or angular.js. (2) has passing knowledge about data mining and machine learning as well as natural language processing; (3) be good at using data analysis packages such as scikit-learn.

Reading Reference for Lecture 2 & 3



Remind -- Apache Hadoop

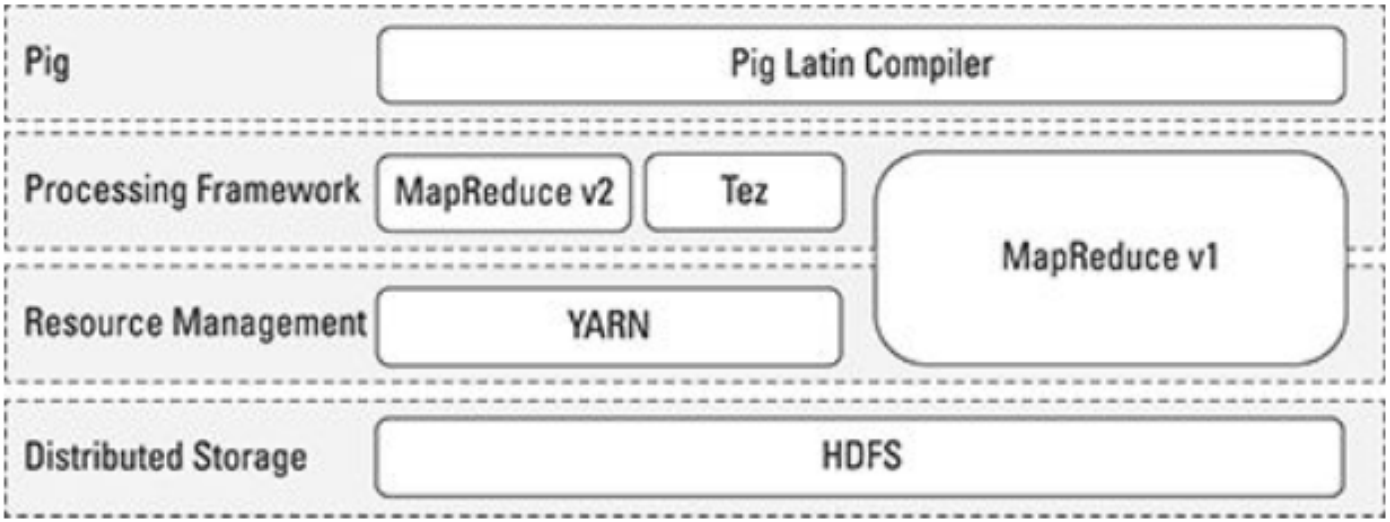
The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing.

The project includes these modules:

- **Hadoop Common**: The common utilities that support the other Hadoop modules.
- **Hadoop Distributed File System (HDFS™)**: A distributed file system that provides high-throughput access to application data.
- **Hadoop YARN**: A framework for job scheduling and cluster resource management.
- **Hadoop MapReduce**: A YARN-based system for parallel processing of large data sets.

http://hadoop.apache.org

Pig



At its core, Pig Latin is a *dataflow* language, where you define a data stream and a series of transformations that are applied to the data as it flows through your application. This is in contrast to a *control flow* language (like C or Java), where you write a series of instructions. In control flow languages, we use constructs like loops and conditional logic (like an if statement). You won't find loops and if statements in Pig Latin.

Pig example

```
input_lines = LOAD '/tmp/my-copy-of-all-pages-on-internet' AS (line:chararray);

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
words = FOREACH input_lines GENERATE FLATTEN(TOKENIZE(line)) AS word;

-- filter out any words that are just white spaces
filtered_words = FILTER words BY word MATCHES '\\w+';

-- create a group for each word
word_groups = GROUP filtered_words BY word;

-- count the entries in each group
word_count = FOREACH word_groups GENERATE COUNT(filtered_words) AS count, group AS word;

-- order the records by count
ordered_word_count = ORDER word_count BY count DESC;
STORE ordered_word_count INTO '/tmp/number-of-words-on-internet';
```

Characteristics of Pig

Most Pig scripts start with the LOAD statement to read data from HDFS. In this case, we're loading data from a .csv file. Pig has a data model it uses, so next we need to map the file's data model to the Pig data mode. This is accomplished with the help of the USING statement. (More on the Pig data model in the next section.) We then specify that it is a comma-delimited file with the PigStorage(',') statement followed by the AS statement defining the name of each of the columns.

Aggregations are commonly used in Pig to summarize data sets. The GROUP statement is used to aggregate the records into a single record mileage_recs. The ALL statement is used to aggregate all tuples into a single group. Note that some statements — including the following SUM statement — requires a preceding GROUP ALL statement for global sums.

FOREACH ... GENERATE statements are used here to transform columns data. In this case, we want to count the miles traveled in the records_Distance column. The SUM statement computes the sum of the record_Distance column into a single-column collection total_miles.

The DUMP operator is used to execute the Pig Latin statement and display the results on the screen.DUMP is used in interactive mode, which means that the statements are executable immediately and the results are not saved. Typically, you will either use the DUMP or STORE operators at the end of your Pig script.

Pig vs. SQL

In comparison to SQL, Pig

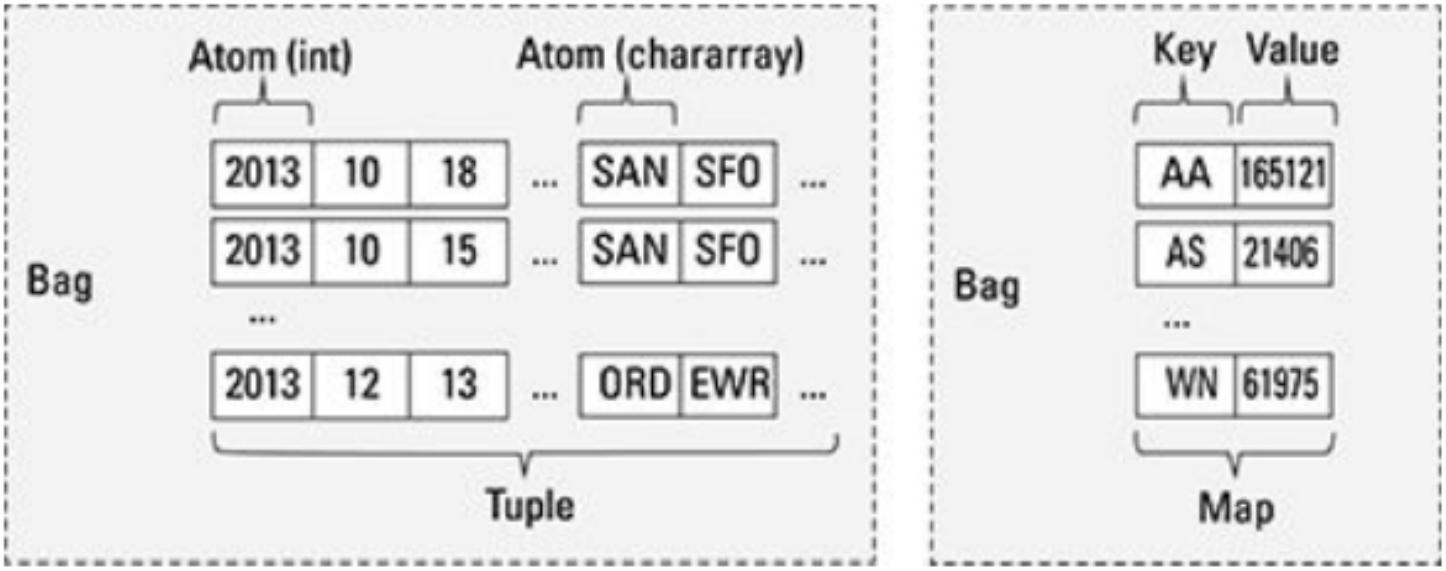
1. uses lazy evaluation,
2. uses ETL,
3. is able to store data at any point during a pipeline,
4. declares execution plans,
5. supports pipeline splits.

On the other hand, it has been argued DBMSs are substantially faster than the MapReduce system once the data is loaded, but that loading the data takes considerably longer in the database systems. It has also been argued RDBMSs offer out of the box support for column-storage, working with compressed data, indexes for efficient random data access, and transaction- level fault tolerance.

Pig Latin is procedural and fits very naturally in the pipeline paradigm while SQL is instead declarative. In SQL users can specify that data from two tables must be joined, but not what join implementation to use. Pig Latin allows users to specify an implementation or aspects of an implementation to be used in executing a script in several ways.

Pig Latin programming is similar to specifying a query execution plan.

Pig Data Types and Syntax



Atom: An atom is any single value, such as a string or number

Tuple: A tuple is a record that consists of a sequence of fields. Each field can be of any type.

Bag: A bag is a collection of non-unique tuples.

Map: A map is a collection of key value pairs.

Pig Latin Operators

Operation	*Operator*	*Explanation*
Data Access	LOAD/STORE	Read and Write data to file system
	DUMP	Write output to standard output (stdout)
	STREAM	Send all records through external binary
	FOREACH	Apply expression to each record and output one or more records
	FILTER	Apply predicate and remove records that don't meet condition
	GROUP/COGROUP	Aggregate records with the same key from one or more inputs
	JOIN	Join two or more records based on a condition
Transformations	CROSS	Cartesian product of two or more inputs
	ORDER	Sort records based on key
	DISTINCT	Remove duplicate records
	UNION	Merge two data sets
	SPLIT	Divide data into two or more bags based on predicate
	LIMIT	subset the number of records

Expressions

In Pig Latin, expressions are language constructs used with the FILTER, FOREACH, GROUP, and SPLIT operators as well as the eval functions.

Expressions are written in conventional mathematical infix notation and are adapted to the UTF-8 character set. Depending on the context, expressions can include:

- Any Pig data type (simple data types, complex data types)
- Any Pig operator (arithmetic, comparison, null, boolean, dereference, sign, and cast)
- Any Pig built-in function.
- Any user-defined function (UDF) written in Java.

In Pig Latin,

- An arithmetic expression could look like this:

```
X = GROUP A BY f2*f3;
```

- A string expression could look like this, where a and b are both chararrays:

```
X = FOREACH A GENERATE CONCAT(a,b);
```

- A boolean expression could look like this:

```
X = FILTER A BY (f1==8) OR (NOT (f2+f3 > f1));
```

Pig User-Defined Functions (UDFs)

```
-- myscript.pig
REGISTER myudfs.jar;
A = LOAD 'student_data' AS (name: chararray, age: int, gpa: float);
B = FOREACH A GENERATE myudfs.UPPER(name);
DUMP B;
```

Command to run the script.

```
java -cp pig.jar org.apache.pig.Main -x local myscript.pig
```

The .java contains UPPER

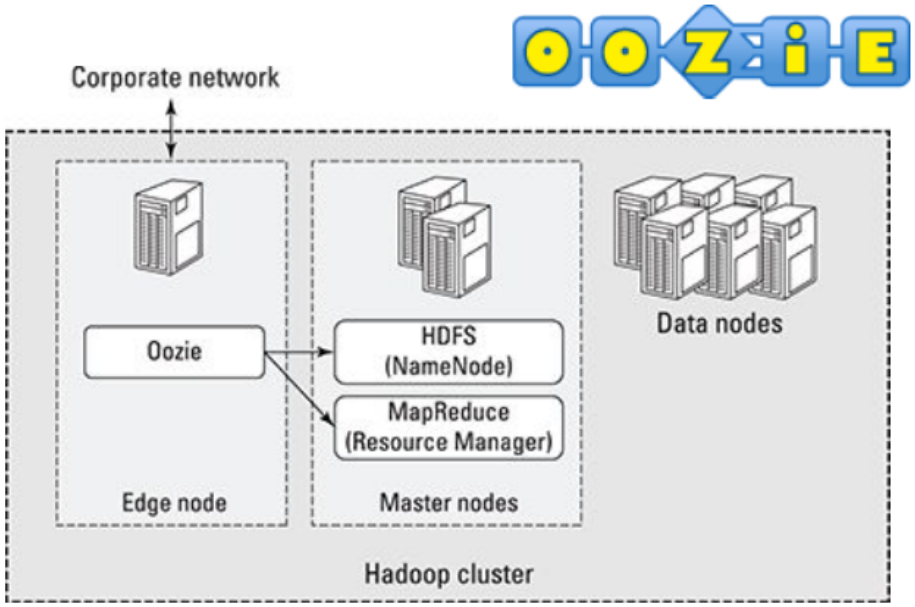
```
package myudfs;
import java.io.IOException;
import org.apache.pig.EvalFunc;
import org.apache.pig.data.Tuple;
import org.apache.pig.impl.util.WrappedIOException;

public class UPPER extends EvalFunc (String)
{
    public String exec(Tuple input) throws IOException {
        if (input == null || input.size() == 0)
            return null;
        try{
            String str = (String)input.get(0);
            return str.toUpperCase();
        }catch(Exception e){
            throw WrappedIOException.wrap("Caught exception processing input row ", e);
        }
    }
}
```

https://pig.apache.org/docs/r0.7.0/udf.html

Oozie Workflow Scheduler for Hadoop

- Oozie supports a wide range of job types, including Pig, Hive, and MapReduce, as well as jobs coming from Java programs and Shell scripts.




```
<workflow-app name="SampleWorkflow"
xmlns="uri:oozie:workflow:0.1">
 <start to="firstJob"/>
 <action name="firstJob">
  <pig>...</pig>
  <ok to="secondJob"/>
  <error to="kill"/>
 </action>
 <action name="secondJob">
  <map-reduce>...</map-reduce>
  <ok to="end" />
  <error to="kill" />
 </action>
 <end name="end"/>
 <kill name="kill">
  <message>"Killed job."</message>
 </kill>
</workflow-app>
```

Sample Oozie XML file

Schedule Workflow by Time

```
<coordinator-app name="sampleCoordinator"
          frequency="${coord:days(1)}"
          start="2014-06-01T00:01Z "
          end="2014-06-01T01:00Z "
          timezone="UTC"
          xmlns="uri:oozie:coordinator:0.1">
 <controls>...</controls>
 <action>

    <workflow>
      <app-path>${workflowAppPath}</app-path>
    </workflow>
  </action>
 </coordinator-app>
```

Schedule Workflow by Time and Data Availability

```
<coordinator-app name="sampleCoordinator"
        frequency="${coord:days(1)}"
        start="${startTime}"
        end="${endTime}"
        timezone="${timeZoneDef}"
        xmlns="uri:oozie:coordinator:0.1">
 <controls>...</controls>
 <datasets>
  <dataset name="input" frequency="${coord:days(1)}" initial-instance="${startTime}" timezone="${timeZoneDef}">
    <uri-template>${triggerDatasetDir}</uri-template>
  </dataset>
 </datasets>
 <input-events>
    <data-in name="sampleInput" dataset="input">
    <instance>${startTime}</instance>
  </data-in>
 </input-events>
 <action>
  <workflow>
    <app-path>${workflowAppPath}</app-path>
  </workflow>
 </action>
</coordinator-app>
```

HBase is modeled after Google's BigTable and written in Java. It is developed on top of HDFS.

It provides a fault-tolerant way of storing large quantities of **sparse data** (small amounts of information caught within a large collection of empty or unimportant data, such as finding the 50 largest items in a group of 2 billion records, or finding the non-zero items representing less than 0.1% of a huge collection).

HBase features compression, in-memory operation, and Bloom filters on a per-column basis

An HBase system comprises a set of tables. Each table contains rows and columns, much like a traditional database. Each table must have an element defined as a Primary Key, and all access attempts to HBase tables must use this Primary Key. An HBase column represents an attribute of an object

Characteristics of data in HBase

Columbia University

Sparse data

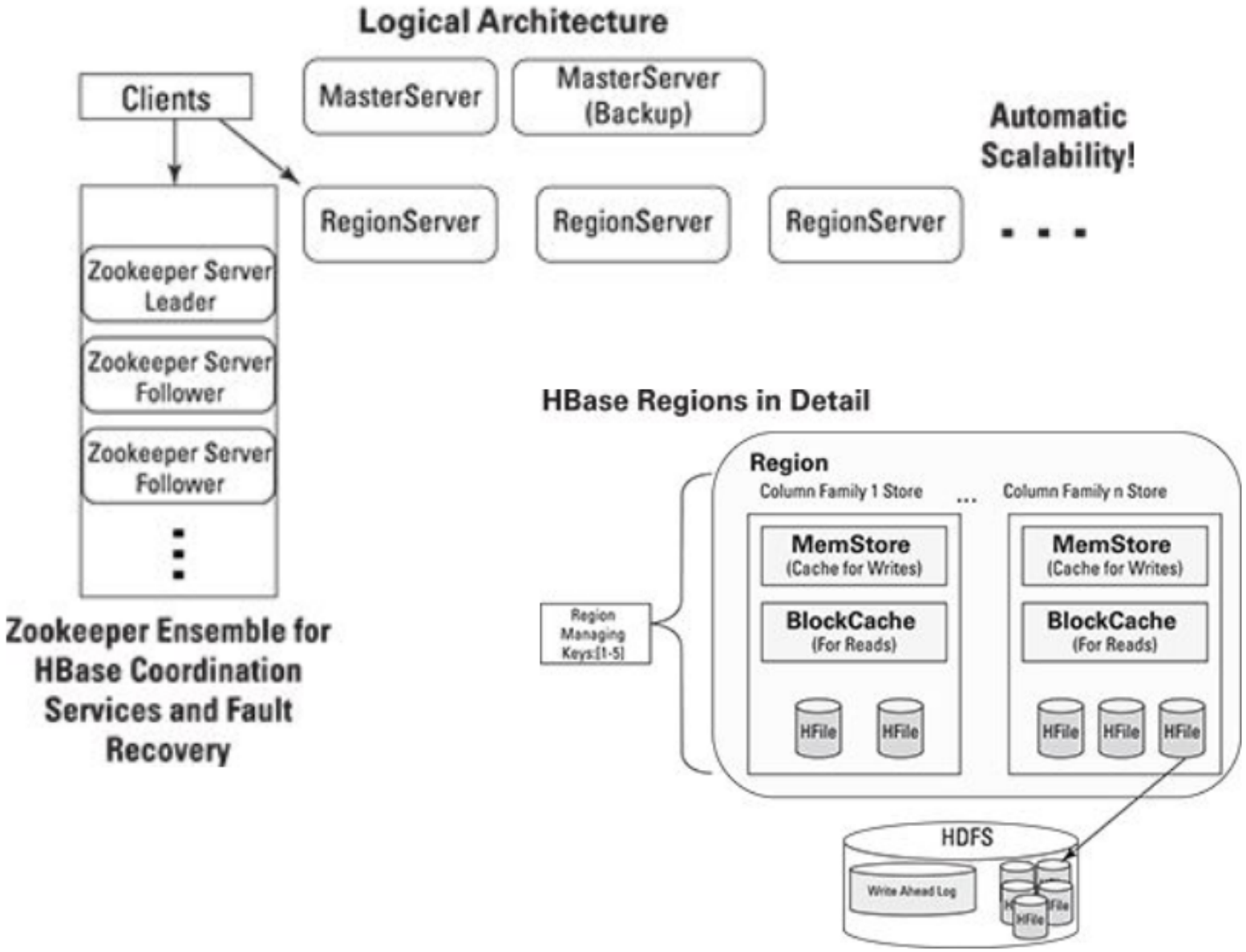
Table 12-1 Traditional Customer Contact Information Table

Customer ID	*Last Name*	*First Name*	*Middle Name*	*E-mail Address*	*Street Address*
00001	Smith	John	Timothy	John.Smith@xyz.com	1 Hadoop Lane, NY 11111
00002	Doe	Jane	NULL	NULL	7 HBase Ave, CA 22222

Row Key	*Column Family: {Column Qualifier:Version:Value}*
00001	CustomerName: {'FN': 1383859182496:'John', 'LN': 1383859182858:'Smith', 'MN': 1383859183001:'Timothy', 'MN': 1383859182915:'T'}
	ContactInfo: {'EA': 1383859183030:'John.Smith@xyz.com', 'SA': 1383859183073:'1 Hadoop Lane, NY 11111'}
00002	CustomerName: {'FN': 1383859183103:'Jane', 'LN': 1383859183163:'Doe',
	ContactInfo: { 'SA': 1383859185577:'7 HBase Ave, CA 22222'}

HDFS lacks **random read and write access**. This is where HBase comes into picture. It's a **distributed, scalable, big data store**, modeled after Google's BigTable. It stores data as key/value pairs.

HBase Architecture



Columbia University

Creating a table

```
hbase(main):002:0> create 'CustomerContactInfo', 'CustomerName', 'ContactInfo'
0 row(s) in 1.2080 seconds
```

COLUMBIA UNIVERSITY

Entering Records

```
hbase(main):008:0> put 'CustomerContactInfo', '00001', 'CustomerName:FN', 'John'
0 row(s) in 0.2870 seconds

hbase(main):009:0> put 'CustomerContactInfo', '00001', 'CustomerName:LN', 'Smith'
0 row(s) in 0.0170 seconds

hbase(main):010:0> put 'CustomerContactInfo', '00001', 'CustomerName:MN', 'T'
0 row(s) in 0.0070 seconds

hbase(main):011:0> put 'CustomerContactInfo', '00001', 'CustomerName:MN', 'Timothy'
0 row(s) in 0.0050 seconds

hbase(main):012:0> put 'CustomerContactInfo', '00001', 'ContactInfo:EA', 'John.Smith@xyz.com'
0 row(s) in 0.0170 seconds

hbase(main):013:0> put 'CustomerContactInfo', '00001', 'ContactInfo:SA', '1 Hadoop Lane, NY 11111'
0 row(s) in 0.0030 seconds

hbase(main):014:0> put 'CustomerContactInfo', '00002', 'CustomerName:FN', 'Jane'
0 row(s) in 0.0290 seconds

hbase(main):015:0> put 'CustomerContactInfo', '00002', 'CustomerName:LN', 'Doe'
0 row(s) in 0.0090 seconds

hbase(main):016:0> put 'CustomerContactInfo', '00002', 'ContactInfo:SA', '7 HBase Ave, CA 22222'
0 row(s) in 0.0240 seconds
```

HBase Example -- III

Scan Results

```
hbase(main):020:0> scan 'CustomerContactInfo', {VERSIONS => 2}
ROW        COLUMN+CELL
 00001     column=ContactInfo:EA, timestamp=1383859183030, value=John.Smith@xyz.com
 00001     column=ContactInfo:SA, timestamp=1383859183073, value=1 Hadoop Lane, NY 11111
 00001     column=CustomerName:FN, timestamp=1383859182496, value=John
 00001     column=CustomerName:LN, timestamp=1383859182858, value=Smith
 00001     column=CustomerName:MN, timestamp=1383859183001, value=Timothy
 00001     column=CustomerName:MN, timestamp=1383859182915, value=T
 00002     column=ContactInfo:SA, timestamp=1383859185577, value=7 HBase Ave, CA 22222
 00002     column=CustomerName:FN, timestamp=1383859183103, value=Jane
 00002     column=CustomerName:LN, timestamp=1383859183163, value=Doe
2 row(s) in 0.0520 seconds
```

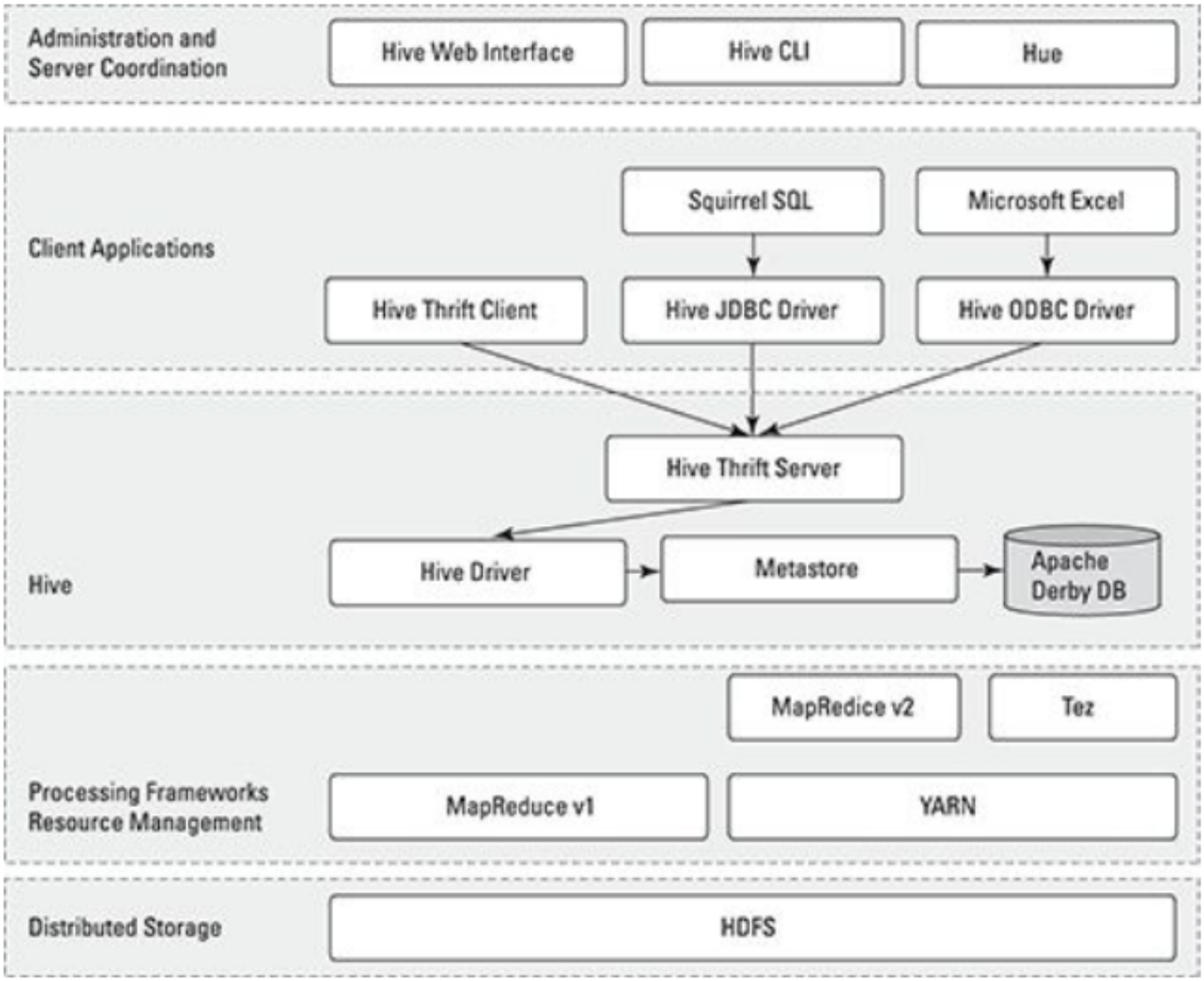
Using the *get* Command to Retrieve Entire Rows and Individual Values

```
(1) hbase(main):037:0> get 'CustomerContactInfo', '00001'
COLUMN                CELL
ContactInfo:EA        timestamp=1383859183030, value=John.Smith@xyz.com
ContactInfo:SA        timestamp=1383859183073, value=1 Hadoop Lane, NY 11111
CustomerName:FN       timestamp=1383859182496, value=John
CustomerName:LN       timestamp=1383859182858, value=Smith
CustomerName:MN       timestamp=1383859183001, value=Timothy
5 row(s) in 0.0150 seconds

(2) hbase(main):038:0> get 'CustomerContactInfo', '00001',
        {COLUMN => 'CustomerName:MN'}
COLUMN                CELL
CustomerName:MN       timestamp=1383859183001, value=Timothy
1 row(s) in 0.0090 seconds

(3) hbase(main):039:0> get 'CustomerContactInfo', '00001',
        {COLUMN => 'CustomerName:MN',
        TIMESTAMP => 1383859182915}
COLUMN                CELL
CustomerName:MN       timestamp=1383859182915, value=T
1 row(s) in 0.0290 seconds
```

Apache Hive



Using Hive to Create a Table

Columbia University

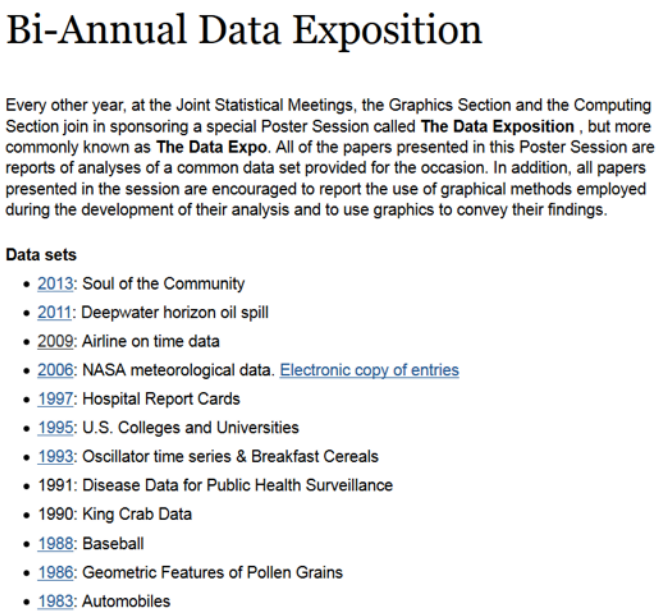
```
 (A) $ $HIVE_HOME/bin hive --service cli
(B) hive> set hive.cli.print.current.db=true;
(C) hive (default)> CREATE DATABASE ourfirstdatabase;
OK
Time taken: 3.756 seconds
(D) hive (default)> USE ourfirstdatabase;
OK
Time taken: 0.039 seconds
(E) hive (ourfirstdatabase)> CREATE TABLE our_first_table (
                > FirstName     STRING,
                > LastName      STRING,
                > EmployeeId     INT);
OK
Time taken: 0.043 seconds
hive (ourfirstdatabase)> quit;
(F) $ ls /home/biadmin/Hive/warehouse/ourfirstdatabase.db
our_first_table
```

Another Hive Example

```
(A) CREATE TABLE IF NOT EXISTS FlightInfo2007 (
Year SMALLINT, Month TINYINT, DayofMonth TINYINT, DayOfWeek TINYINT,
DepTime SMALLINT, CRSDepTime SMALLINT, ArrTime SMALLINT, CRSArrTime SMALLINT,
UniqueCarrier STRING, FlightNum STRING, TailNum STRING,
ActualElapsedTime SMALLINT, CRSElapsedTime SMALLINT,
AirTime SMALLINT, ArrDelay SMALLINT, DepDelay SMALLINT,
Origin STRING, Dest STRING,Distance INT,
TaxiIn SMALLINT, TaxiOut SMALLINT, Cancelled SMALLINT,
CancellationCode STRING, Diverted SMALLINT,
CarrierDelay SMALLINT, WeatherDelay SMALLINT,
NASDelay SMALLINT, SecurityDelay SMALLINT, LateAircraftDelay SMALLINT)
COMMENT 'Flight InfoTable'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE
TBLPROPERTIES ('creator'='Bruce Brown', 'created_at'='Thu Sep 19 10:58:00 EDT 2013');
```

Homework #1

1. Install Hadoop
2. Download Airline Data and one your own selected dataset from Stat-Computing.org
3. Learn to use PIG. You can try the example in the reference
4. Use Oozie to schedule a few jobs
5. Try HBase. Use your own example
6. Try Hive. Use your own example



Bi-Annual Data Exposition

Every other year, at the Joint Statistical Meetings, the Graphics Section and the Computing Section join in sponsoring a special Poster Session called **The Data Exposition** , but more commonly known as **The Data Expo**. All of the papers presented in this Poster Session are reports of analyses of a common data set provided for the occasion. In addition, all papers presented in the session are encouraged to report the use of graphical methods employed during the development of their analysis and to use graphics to convey their findings.

Data sets

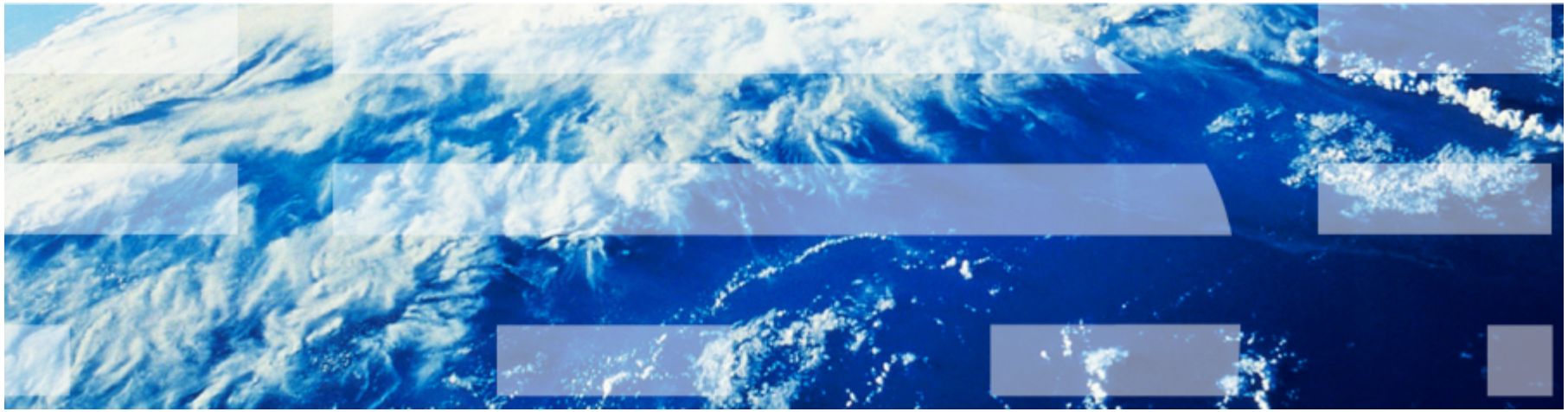
- 2013: Soul of the Community
- 2011: Deepwater horizon oil spill
- 2009: Airline on time data
- 2006: NASA meteorological data. Electronic copy of entries
- 1997: Hospital Report Cards
- 1995: U.S. Colleges and Universities
- 1993: Oscillator time series & Breakfast Cereals
- 1991: Disease Data for Public Health Surveillance
- 1990: King Crab Data
- 1988: Baseball
- 1986: Geometric Features of Pollen Grains
- 1983: Automobiles

http://stat-computing.org/dataexpo/

Questions?

E6893 Big Data Analytics:

Demo Session for HW I



Homework #1

1. Install Hadoop
2. Download Airline Data and one your own selected dataset from Stat-Computing.org
3. Learn to use PIG. You can try the example in the reference
4. Use Oozie to schedule a few jobs
5. Try HBase. Use your own example
6. Try Hive. Use your own example

Bi-Annual Data Exposition

Every other year, at the Joint Statistical Meetings, the Graphics Section and the Computing Section join in sponsoring a special Poster Session called **The Data Exposition** , but more commonly known as **The Data Expo**. All of the papers presented in this Poster Session are reports of analyses of a common data set provided for the occasion. In addition, all papers presented in the session are encouraged to report the use of graphical methods employed during the development of their analysis and to use graphics to convey their findings.

Data sets

- 2013: Soul of the Community
- 2011: Deepwater horizon oil spill
- 2009: Airline on time data
- 2006: NASA meteorological data. Electronic copy of entries
- 1997: Hospital Report Cards
- 1995: U.S. Colleges and Universities
- 1993: Oscillator time series & Breakfast Cereals
- 1991: Disease Data for Public Health Surveillance
- 1990: King Crab Data
- 1988: Baseball
- 1986: Geometric Features of Pollen Grains
- 1983: Automobiles

http://stat-computing.org/dataexpo/

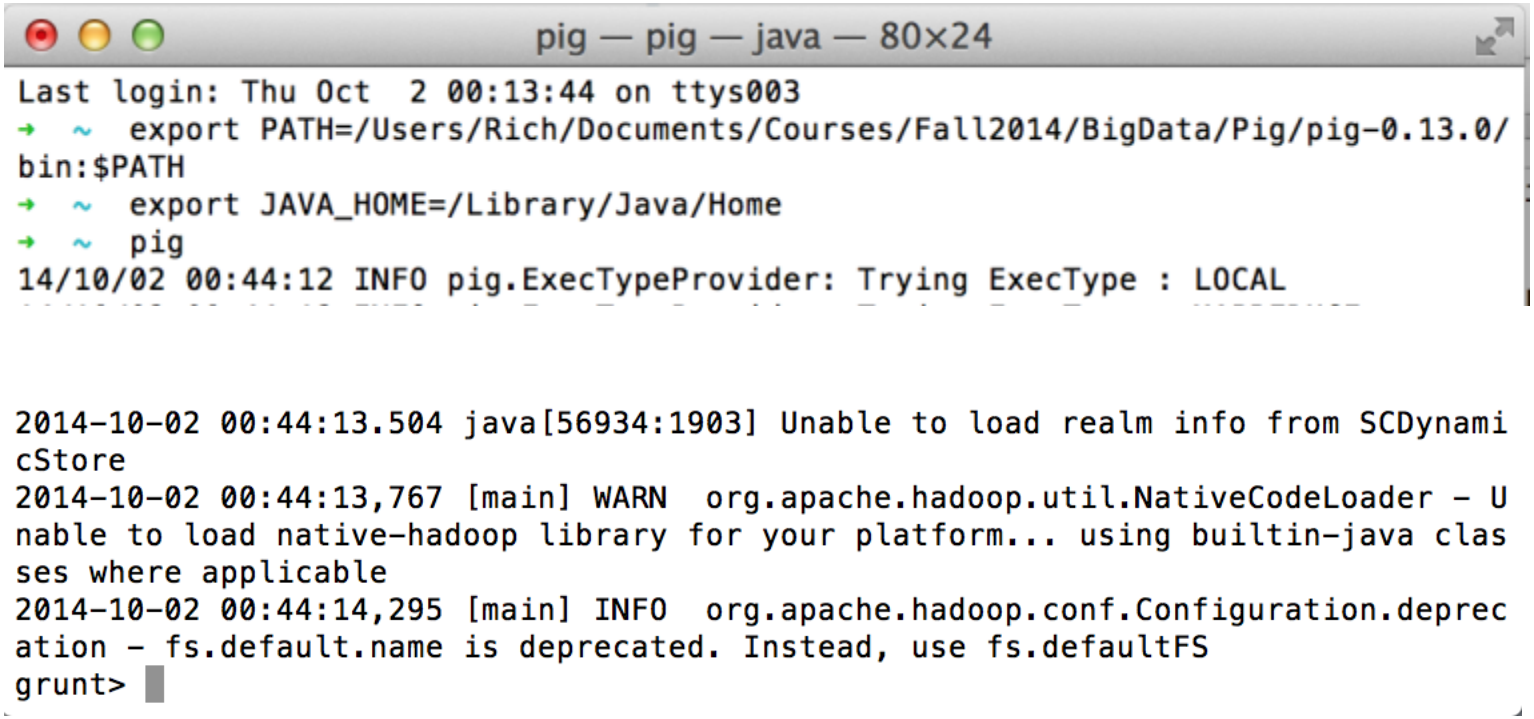
Part I: Pig installation and Demo

Pig is a platform for **analyzing large data sets** that consists of a **high-level language** for expressing data analysis programs, coupled with infrastructure for evaluating these programs.

1. Installation of Pig:
https://pig.apache.org/docs/r0.7.0/setup.html

Download pig and run following sentence:

export PATH=/<my-path-to-pig>/pig-n.n.n/bin:$PATH



2. Running pig in local mode:

```
pig -x local

movies = LOAD '/Users/Rich/
Documents/Courses/Fall2014/BigData/
Pig/movies_data.csv' USING
PigStorage(',') as
(id,name,year,rating,duration);

DUMP movies
```

Filter:

List the movies that were released between 1950 and 1960

```
movies_1950_1960 = FILTER movies BY (float)year>1949 and (float)year<1961;

store movies_1950_1960 into '/Users/Rich/Desktop/Demo/movies_1950_1960';
```

Foreach Generate:

List movie names and their duration (in minutes)

```
movies_name_duration = foreach movies
generate name, (float)duration/3600;

store movies_name_duration into '/Users/Rich/
Desktop/Demo/movies_name_duration';
```

Order:

List all movies in descending order of year

movies_year_sort =**order** movies by year **desc**;

store movies_year_sort into '/Users/Rich/Desktop/Demo/movies_year_sort';

3. Running pig with HDFS:

pig

Run HDFS first:

ssh localhost

cd /usr/local/Cellar/hadoop/2.5.0

sbin/start-dfs.sh

3. Upload file to HDFS:

Make a directory:

```
bin/hdfs dfs -mkdir /PigSource
```

Upload a file:

```
bin/hdfs dfs -put /Users/Rich/Documents/
Courses/Spring2014/CloudComputing/HW/
MINI5/movies_data.csv /PigSource
```

4. Run pig in grunt with HDFS:

pig

```
movies = LOAD '/PigSource/
movies_data.csv' USING PigStorage(',')
as (id,name,year,rating,duration);

DUMP movies
```

5. Run .pig file with HDFS:

pig

pig /Users/Rich/Documents/Courses/Fall2014/BigData/Pig/run1.pig

Part II: Hbase installation and Demo

HBase is an open source, non-relational, distributed database modeled after Google's Big Table and written in Java.

It runs on top of HDFS, and can serve as input and output for MapReduce jobs run in Hadoop.

Access through Java API and Pig.

Hbase Configuration:

Download and configure Hbase by following the instruction online.

To run Hbase, under your Hbase path:

$bin/start-hbase.sh

Enter shell

$bin/hbase shell

Also visit localhost:60010 to check Hbase webUI

Hbase Command:

Create:

hbase>create ‘table’, ‘cf’

hbase>put ’table', ’r1', ’cf:c1', 'value1’

hbase>scan ‘table’

Also we can do count, get, delete and drop. etc much like other DB systems.

Part III: Hive installation and Demo

Install steps:

Mac: you can first install brew

```
$ brew install Hive
```

Linux: cd ~/Downloads

```
wget http://mirror.cc.columbia.edu/pub/software/
apache/hive/
hive-0.13.1/apache-hive-0.13.1-bin.tar.gz
cp apache-hive-0.13.1-bin.tar.gz ~
cd ~
tar zxvf apache-hive-0.13.1-bin.tar.gz
mv apache-hive-0.13.1-bin hive
cd hive
cd bin
./hive
```

Simple demo for Hive

Step 1. start Hadoop, create a file under /user/yourname

```
hadoop fs -mkdir test
```

Step 2. put your dataset into hdfs

```
hadoop fs -put (your dataset path) test
hadoop fs -ls test
```

```
Found 1 items
-rw-r--r--   1 huanglin supergroup        204 2014-09-30 22:49 test/client.txt
```

Simple demo for Hive

Step 3. create the table (SQL) in Hive shell

```
hive
```

```
hive> create table client(
    > name String,
    > gender String,
    > age INT,
    > job String,
    > nation String,
    > tele INT)
    > row format delimited
    > fields terminated by '\t'
    > lines terminated by '\n'
    > stored as textfile;
OK
Time taken: 0.049 seconds
```

```
hive> select * from client;
OK
John    male    17      students        USA     111111
Henry   male    25      sportsman       UK      222222
Kim     female  29      actor   Korea   33333
Zhu     female  21      graduate        China   444444
Alex    male    47      professor       France  555555
Luca    male    30      banker  Swiss   666666
Time taken: 0.038 seconds, Fetched: 6 row(s)
```

Step 4. store the data into table

```
load data inpath '/user/yourname/test' into table client;
```

You can see your data has already been put into the table

Simple demo for Hive

1. select * from client where name='Henry';

```
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Starting Job = job_1412113170187_0003, Tracking URL = http://dyn-160-39-231-29.d
yn.columbia.edu:8088/proxy/application_1412113170187_0003/
Kill Command = /usr/local/Cellar/hadoop/2.5.1/libexec/bin/hadoop job  -kill job_
1412113170187_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 0
2014-09-30 22:53:03,624 Stage-1 map = 0%,  reduce = 0%
2014-09-30 22:53:09,947 Stage-1 map = 100%,  reduce = 0%
Ended Job = job_1412113170187_0003
MapReduce Jobs Launched:
Job 0: Map: 1   HDFS Read: 417 HDFS Write: 34 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Henry   male    25      sportsman       UK      222222
Time taken: 15.673 seconds, Fetched: 1 row(s)
```

2. select avg(age) from client where gender='male';

```
OK
29.75
Time taken: 21.521 seconds, Fetched: 1 row(s)
```

Simple demo for Hive

Use JDBC(JAVA) to access data in Hive Server

Other choice: Python, PHP etc.

```
Connection con = DriverManager.getConnection(
                  "jdbc:hive://localhost:10000/default", "", "");
Statement stmt = con.createStatement();
String tableName = "client1";
stmt.execute("drop table if exists " + tableName);
stmt.execute("create table " + tableName + " (name String, gender String, age INT, job String, "
        + "nation String, tele INT) row format delimited fields terminated by '\t' lines "
        + "terminated by '\n' stored as textfile");
System.out.println("Create table success!");
String filepath = "/user/huanglin/hive";
String sql = "load data inpath '" + filepath + "' into table " + tableName;
System.out.println("Running: " + sql);
ResultSet res = stmt.executeQuery(sql);
sql = "select name,nation from " + tableName + " where age>25";
System.out.println("Running: " + sql);
res = stmt.executeQuery(sql);
while (res.next()) {
    System.out.println(res.getString(1) + "\t"
                                        + res.getString(2));
}
sql = "select * from " + tableName + " where nation='USA'";
System.out.println("Running: " + sql);
res = stmt.executeQuery(sql);
while (res.next()) {
    System.out.println(res.getString(1) + "  "+ res.getString(2) + "  "+ String.valueOf(res.getInt(3))+
            "  "+res.getString(4)+"  "+res.getString(5)+"  "+String.valueOf(res.getInt(6)));
}
```

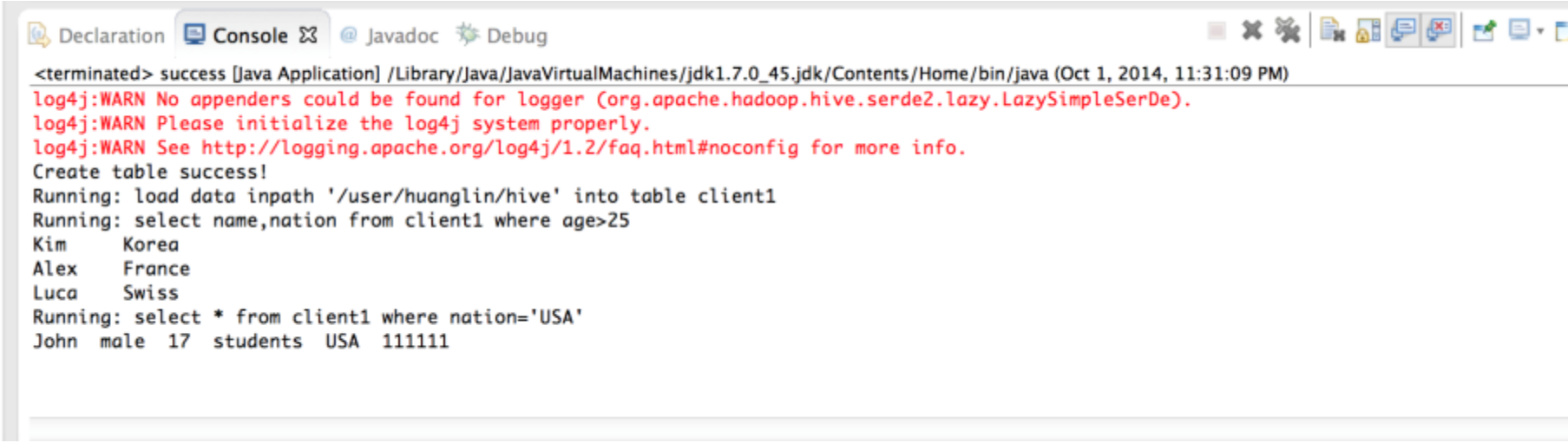
Simple demo for Hive

Open the Hive (Starting Hive Thrift Server)

```
serverhive --service hiveserver -p 10000
```

URL: jdbc:hive://localhost:10000/default

Result in the console



Simple demo for Hive

Example. Jetty Server with Jersey, restful web service

Create simple API (Http request, XML/JSON format output)

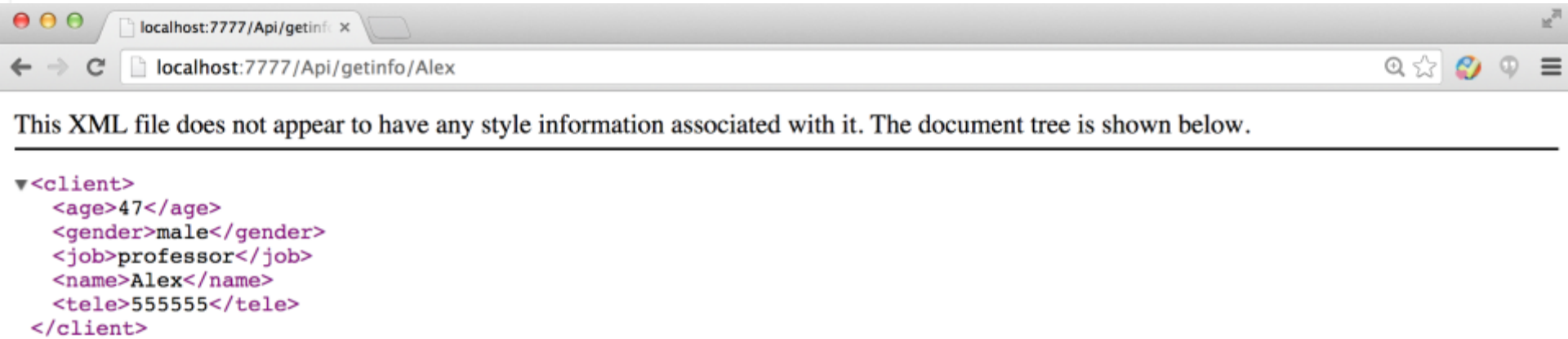
```
@Path("/Api")
public class service {
 @GET
 @Path("/getinfo/{name}")
 @Produces(MediaType.APPLICATION_XML)
 public client getinfo(@PathParam("name") String name) throws SQLException {
     try {
         Class.forName("org.apache.hadoop.hive.jdbc.HiveDriver");
     } catch (ClassNotFoundException e) {
         e.printStackTrace();
         System.exit(1);
     }
     Connection con = DriverManager.getConnection(
                       "jdbc:hive://localhost:10000/default", "", "");
     Statement stmt = con.createStatement();
     String tableName = "client1";
     stmt.execute("drop table if exists " + tableName);
     stmt.execute("create table " + tableName + " (name String, gender String, age INT, job String, "
             + "nation String, tele INT) row format delimited fields terminated by '\t' lines "
             + "terminated by '\n' stored as textfile");
     String filepath = "/user/huanglin/hive";
     String sql = "load data inpath '" + filepath + "' into table " + tableName;
     ResultSet res = stmt.executeQuery(sql);
     sql = "select * from client1 where name='"+name+"'";
     System.out.println("Running: " + sql);
     res = stmt.executeQuery(sql);
     while (res.next()) {
         client user = new client(); user.setName(res.getString(1)); user.setGender(res.getString(2));
         user.setAge(res.getInt(3)); user.setJob(res.getString(4)); user.setNation(res.getString(5));
         user.setTele(res.getInt(6)); return user;
     }
  return null;
 }
```

Simple demo for Hive

Example. Jetty Server with Jersey, restful web service

```
public class server {

    public static void main(String[] args) throws Exception {
        Server server = new Server(7777);
        ServletContextHandler context = new ServletContextHandler(ServletContextHandler.SESSIONS);
        context.setContextPath("/");
        server.setHandler(context);
        ServletHolder jerseyServlet = context.addServlet(org.glassfish.jersey.servlet.ServletContainer.class, "/*");
        jerseyServlet.setInitOrder(0);
        jerseyServlet.setInitParameter("jersey.config.server.provider.classnames", "rest.service");
        server.start();
        server.join();
    }
```



Part IV: Oozie installation and Demo

- Before Getting Started
- How to solve
- Virtual Machine Environment Spec
- Run and Test

Before Getting Started

- Oozie isn't compatible with Hadoop 2.
- BigTop came for rescue.
- However, BigTop doesn't supported Hadoop 2 now.

SO HOW DO WE SUPPOSE TO DO ?

Don’t worry, we found the solution

- We have setup a virtual machine environment for you, which already includes Pig, Hadoop and Oozie installed and configured.
- We have already verified that Pig, Hadoop and Oozie can work with each other in our provided virtual environment with no conflict.

Spec

- OS: Linux OS (Ubuntu 14.04)
- Hadoop: 2.5.0 (locate in /usr/local)
- Ooze: 4.0.1 (locate in /usr/local)
- Maven: 3.0.5
- Pig: 0.13 (locate in /usr/local)
- Java: 1.6

NOTE: JAVA_HOME and HADOOP_PREFIX has already setup

How to Run Oozie (1)

(1) (IMPORTANT) SSH to localhost and start HDFS

```
$ ssh localhost
$ cd /usr/local/hadoop-2.5.0
$ ./sbin/start-dfs.sh
$ ./sbin/start-yarn.sh
```

How to Run Oozie (1)

(2) 6 Nodes should be running shown as below:

$ jps

8633 Jps

5118 NameNode

5238 DataNode

5411 SecondaryNameNode

5625 ResourceManager

5750 NodeManager

How to Run Oozie (2)

(3) Start Oozie

$ cd /usr/local/oozie-4.0.1

$ ./bin/oozied.sh start

(4) Check Oozie running status

$ cd /usr/local/oozie-4.0.1

$ ./bin/oozie admin -oozie http://localhost:11000/oozie -status

NORMAL

How to Run Oozie (3)

(5) Untar Oozie example

```
$ cd /usr/local/oozie-4.0.1
$ tar –zxvf oozie-examples.tar.gz
```

(6-1) Change Namenode port number from 8020 to 9000

```
$ cd /usr/local/oozie-4.0.1/examples
$ find ./ -type f -exec sed -i -e 's/8020/9000/g' {} \;
```

(6-2) Change Jobtracker port number from 8021 to 8088

```
$ cd /usr/local/oozie-4.0.1/examples
$ find ./ -type f -exec sed -i -e 's/8021/8088/g' {} \;
```

How to Run Oozie (4)

(7) Submit a job to Oozie

```
$ cd /usr/local/oozie-4.0.1
$ ./oozie job -oozie http://localhost:11000/oozie -config
    examples/apps/map-reduce/job.properties -run
```

[YOUR_JOB_ID] will return here

(8) Check the job status

```
$ cd /usr/local/oozie-4.0.1
$ ./oozie job -oozie http://localhost:11000/oozie -info
    [YOUR_JOB_ID]
```

How to Test Oozie

(1) Check the Oozie log file logs/oozie.log to ensure Oozie started properly.

(2) Using the Oozie command line tool check the status of Oozie:

```
$ cd /usr/local/oozie-4.0.1
$ ./bin/oozie admin –oozie
    http://localhost:11000/oozie  -status
```

(3) Using a browser go to the oozie web console Oozie status should be NORMAL