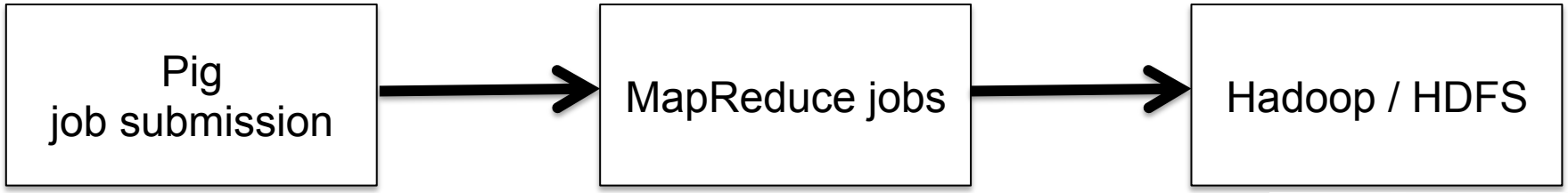
Pig

A high-level programming interface

- Apache Pig is a platform raising a level of abstraction for processing large datasets. Its language, Pig Latin is a simple query algebra expressing data transformations and applying functions to records



- Started at Yahoo! Research, >60%

of Hadoop jobs within Yahoo! are Pig jobs

Motivations

- MapReduce requires a Java programmer
  - Solution was to abstract it and create a system where users are familiar with scripting languages
- Other than very trivial applications, MapReduce requires multiple stages, leading to long development cycles
  - Rapid prototyping. Increased productivity
- In MapReduce users have to reinvent common functionality (join, filter, etc.)
  - Pig provides them

Used for

- Rapid prototyping of algorithms for processing large datasets
- Log analysis
- Ad hoc queries across various large datasets
- Analytics (including through sampling)
- **Pig Mix** provides a set of performance and scalability benchmarks. Currently 1.1 times MapReduce speed.

Using Pig

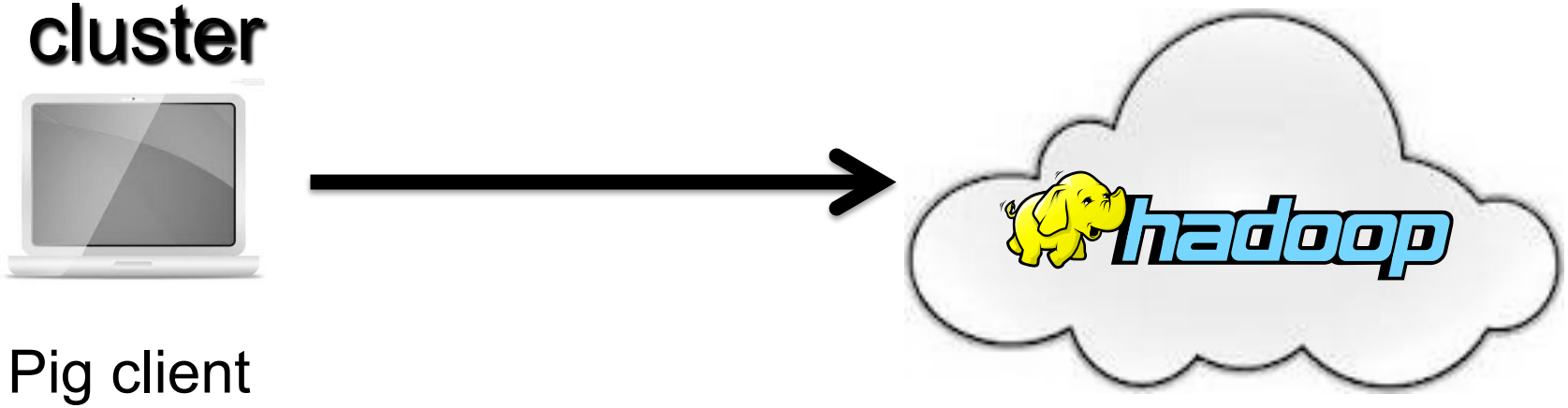
- Grunt, the Pig shell
- Executing scripts directly
- Embedding Pig in Java (using PigServer, similar to SQL using JDBC), or Python
- A range of tools including Eclipse plug-ins
  - PigPen, Pig Editor…

Execution modes

- Pig has two execution types or modes: local mode and Hadoop mode
- *Local*
  - Pig runs in a single JVM and accesses the local filesystem. Starting form v0.7 it uses the Hadoop job runner.
- *Hadoop mode*
  - Pig runs on a Hadoop cluster (you need to tell Pig about the version and point it to your Namenode and Jobtracker

Running Pig

- Pig resides on the user's machine and can be independent from the Hadoop cluster
- Pig is written in Java and is portable
  - Compiles into map reduce jobs and submit them to the cluster
- No need to install anything extra on the cluster



Pig client

How does it work

- Pig defines a **DAG**. A step-by-step set of operations, each performing a transformation
- Pig defines a logical plan for these transformations:

```
A = LOAD 'file' as (line);
B = FOREACH A GENERATE
FLATTEN(TOKENIZE(line)) AS
word;
C = GROUP B BY word;
D = FOREACH C GENERATE
group, COUNT(B);
STORE D INTO 'output'
```

- Parses, checks, & optimises
- Plan the execution
  - Maps & Reduces
- Passes the jar to Hadoop
- Monitor the progress

Operators

- Read/Write operators
  - LOAD:
    - alias = LOAD 'file' [USING function] [AS schema];
  - LIMIT
    - alias = LIMIT alias n;
  - DUMP
    - DUMP alias;
  - STORE
    - STORE alias INTO 'directory' [USING function];
  - PigStorage: parses a line of input into fields using a character delimiter
  - BinStorage: loads and stores data in machine-readable format
  - TextLoader: Loads unstructured data in UTF-8 format
  - …

Read/Write Example

```
grunt> a = LOAD 'A' using PigStorage(',') AS
                                (a1, a2, a3);
grunt> b = LOAD 'B' using PigStorage(',') AS
                                (b1, b2, b3);
grunt> DUMP a;
(0,1,2)
(1,3,4)
grunt> DUMP b;
(0,5,2)
(1,7,8)
```

Data types & expressions

- Scalar type:
  - int, long, float, double, chararray, bytearray
- Complex type representing nested structures:
  - Tuple: sequence of fields of any type
    - Ex: (1.2, apple, 10)
  - Bag: an unordered collection of tuples
    - Ex: {(1.2, apple, 10), (3.2, orange, 5)}
  - Map: a set of key-value pairs. Keys must be atoms, values may be any type

Schemas

- Schemas enable you to associate names and types of the fields in the relation
- Schemas are optional but recommended whenever possible; type declarations result in better parse-time error checking and more efficient code execution
- They are defined using the AS keyword with operators
- Schema definition for simple data types:

```
> records = LOAD 'input/data' AS (id:int, date:chararray);
```

Statements and aliases

- Each statement, defining a data processing operator / relation, produces a dataset with an alias

```
grunt>records = LOAD 'input/data' AS (id:int, date:chararray);
```

- LOAD returns a tuple, which elements can be referenced by name
- Very useful operators (diagnostic) are DUMP, ILLUSTRATE, and DESCRIBE

Filtering data

- FILTER is user to work with tuples and rows of data
- Select data you want, or remove the data you are not interested in
- Filtering early in the processing pipeline minimises the amount of data flowing through the system, which can improve efficiency

```
grunt>filtered_records = FILTER records BY id == 234;
```

Expressions

- Used in Pig as a part of a statement:
  - field name: users, ipaddress, date
  - arithmetic: + - * / %
  - conditional: if , (x ? y : z)
  - comparison: < > >= <= == !=
  - Boolean: and or not
  - Pattern matching: x matches, regex
    - Ex: REGEX_EXTRACT('192.168.1.5:8020', '(.*):(.*)', 1)

      (\\d+\\.\\d+\\.\\d+)
  - etc.

FOREACH … GENERATE

- `FOREACH .. GENERATE` acts on columns on every row in a relation

```
grunt> ids = FOREACH records GENERATE id;
```

- ***Positional reference***. This statement has the same output

```
grunt> ids = FOREACH records GENERATE $0;
```

- The elements of '`ids`' however are not named '`id`' unless you add '`AS id`' at the end of your statement

```
grunt> ids = FOREACH records GENERATE $0 AS id;
```

Grouping and joining

- `GROUP .. BY` makes an output bag containing grouped fields with the same schema using a grouping key
- `JOIN` performs inner of two or more relations based on common field values.
- You can also perform outer joins using keywords `left`, `right` and `full`

Combining, splitting…

- the `UNION` operator to merge the contents of two or more relations
- `SPLIT` partitions a relation into two or more relations

```
grunt> DUMP a;
(0,1,2)
(1,3,4)
grunt> DUMP b;
(0,5,2)
(1,7,8)
```

```
grunt> c = UNION a, b;
grunt> DUMP c;
(0,1,2)
(0,5,2)
(1,3,4)
(1,7,8)
```

```
grunt> SPLIT c INTO d IF $0 == 0, e IF $0 == 1;
grunt> DUMP d;
(0,1,2)
(0,5,2)
grunt> DUMP e;
(1,3,4)
(1,7,8)
```

(Chuck Lam)

Ordering, Sampling…

- `ORDER` imposes an order on the output to sort a relation by one or more fields
- The `LIMIT` statement limits the number of results
- the `SAMPLE` operator selects a random data sample with the stated sample size

```
grunt>sample_records = SAMPLE records 0.01;
```