Hadoop Introduction

Java以外で始めるHadoopによるビッグデータ活用

IT Technical Seminar V13 (2012)

はじめに

- **鈴木 優一（すずきゆういち）**
  - 株式会社オープントーン
    金融ソリューション事業部
    入社４年目

  - C#や.NET Freamwork界隈が好物です。

  - Twitter ID : @U1 Buzz

Agenda

1. Hadoopおさらい
2. HadoopStreaming
3. Hive
4. Demo （Apacheログ解析）
5. まとめ

Hadoopの概要

- **オープンソースの大規模分散フレームワーク**
  - ベースはGoogleが発表した2つの論文
    - The Google File System(ACM,2003)
    - MapReduce: Simplified Data Processing on Large Clusters (OSDI,2004)
  - Doun Cutting氏が中心となりJavaで実装
- **2つの主要コーポネントから構成**
  - HDFS(Hadoop Distributed File System)
    - 分散ファイルシステム（Master/Slave方式で実現する）
  - Hadoop MapReduce Freamework
    - 大量のデータをキーなどの情報で分散して、多数のサーバで並列処理を行う分散処理フレームワーク。
      サーバーを増築することで、処理性能を向上できる。

Hadoopの特徴

- **個別設計不要な分散処理**
  - MapReduceフレームワークにより、プログラム個別に分散処理ロジックを用意することなく、気軽に分散処理を利用することができる。
    ※従来は、プログラムごとに分散処理方式を検討する必要があった。

- **高いスケーラビリティ**
  - サーバ台数を増やすことで、保存するデータサイズ・計算・バッチ処理のI/O性能を柔軟に拡張できる。
  - 特にI/O性能は数台規模でも効果が見込める。

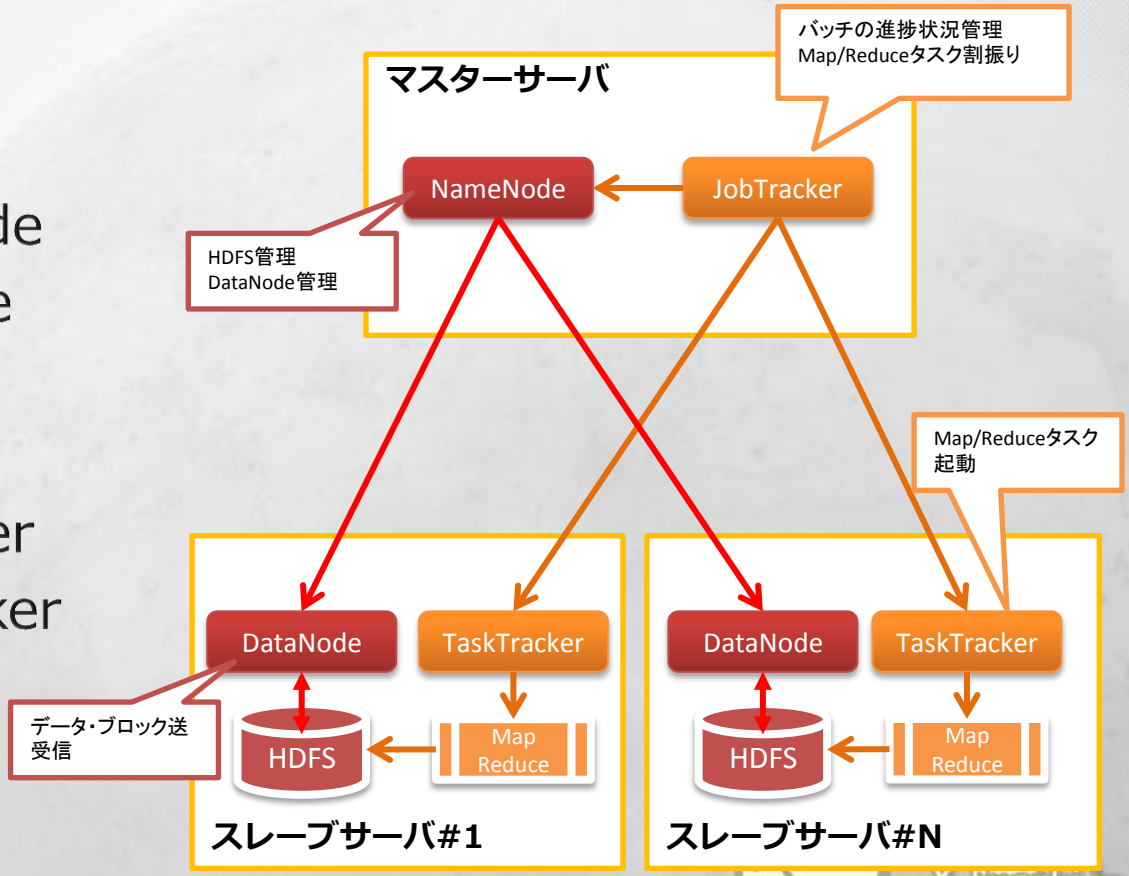
- **コモディティサーバ利用を前提とした設計**
  - Hadoopを構成するサーバに、特別な製品などは不要。
  - 一部のサーバが故障しても、サービスの可用性に影響を与えない仕組み。
    - データは複数分配して配置、処理も異常時はリトライ

Hadoop クラスタ構成

- **集中管理型分散システム**
  - 分散処理ジョブやデータ管理はマスタサーバで実施
  - スレーブサーバは、分散処理の実行やデータの実体を保存
  - スレーブサーバのクラスタへの参加・離脱は自動的

- **HDFS**
  - マスター：NameNode
  - スレーブ：DataNode

- **MapReduce**
  - マスター：JobTracker
  - スレーブ：TaskTracker



Hadoop 適用範囲

- **向いている**
  - 数10GB以上の巨大データを扱うシステム
    - メモリに載ってしまうようなデータだと意味がない。
  - バッチ処理的なアプリケーション
    - リアルタイム性がないもの。もしくはリアルタイム処理の前処理に利用する。

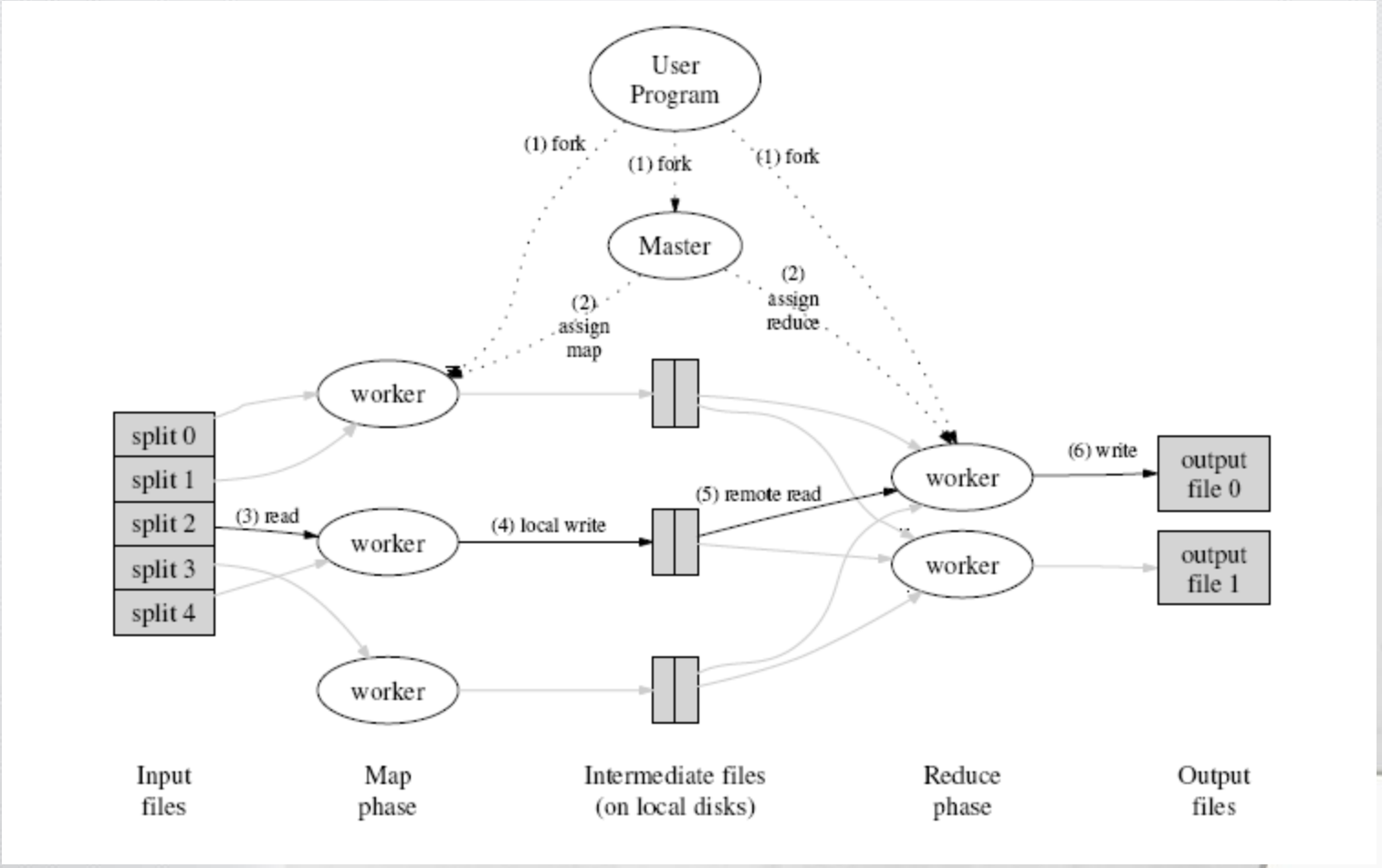
- **利用例**
  - ログ解析（Log Analysis）
  - レコメンデーション（Recommendation）
  - データマイニング（Data Mining）
  - 機械学習（Machine learning）

Hadoop≠RDBMS

- **データ格納**
  - Hadoop：自由に格納、処理するときに意味づけ
  - RDBMS：格納するときに定義、正規化された状態で保存
- **データアクセス**
  - Hadoop：シーケンシャル
  - RDBMS：ランダムアクセス
- **処理**
  - Hadoop：大規模なデータを一気に処理
  - RDBMS：小さいデータを高速に処理

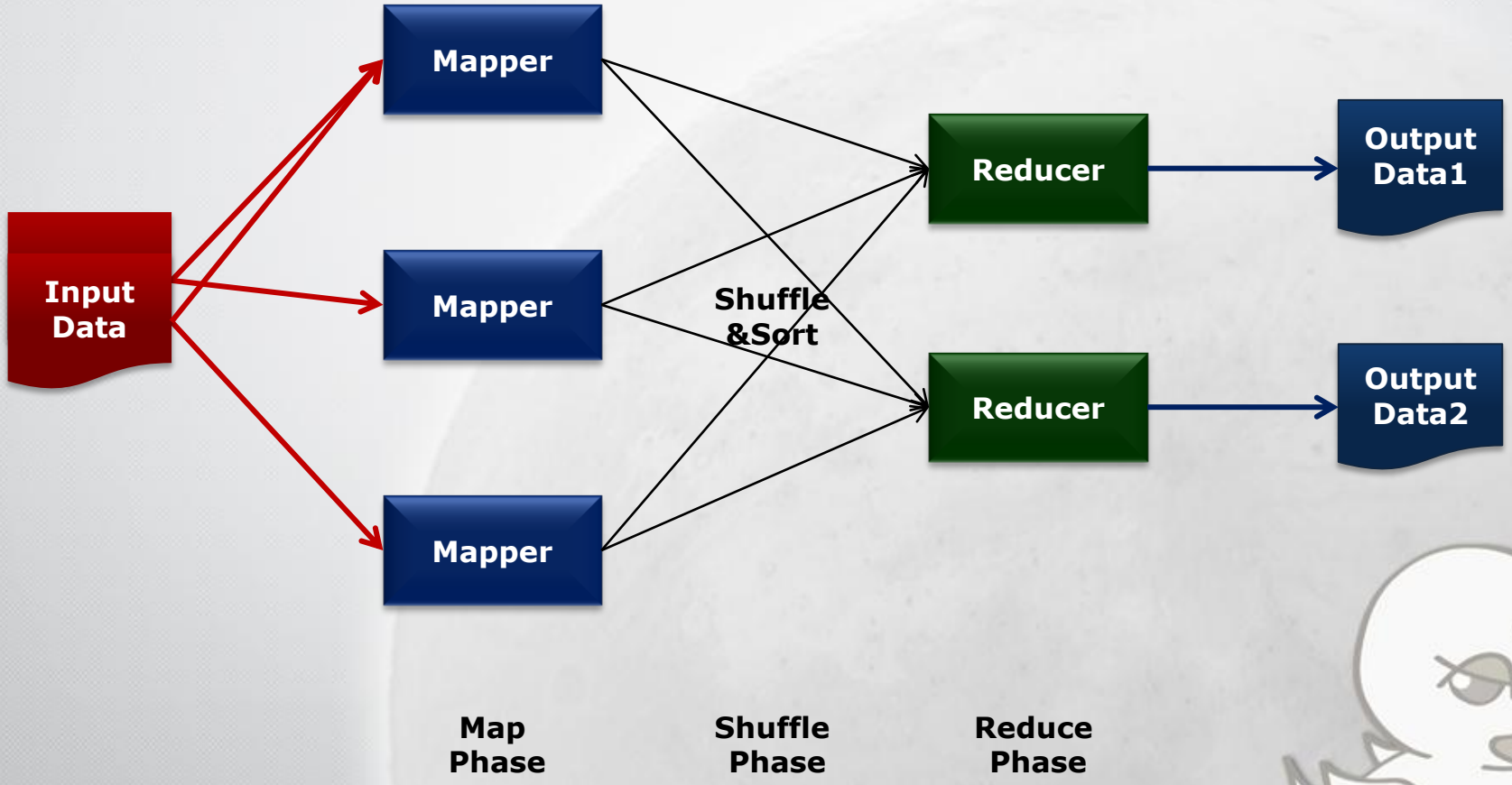
MapReduce

MapReduce ProgrammingModel



Dean, Jeff and Ghemawat, Sanjay. MapReduce: Simplified Data Processing on Large Clusters
http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/archive/mapreduce-osdi04.pdf

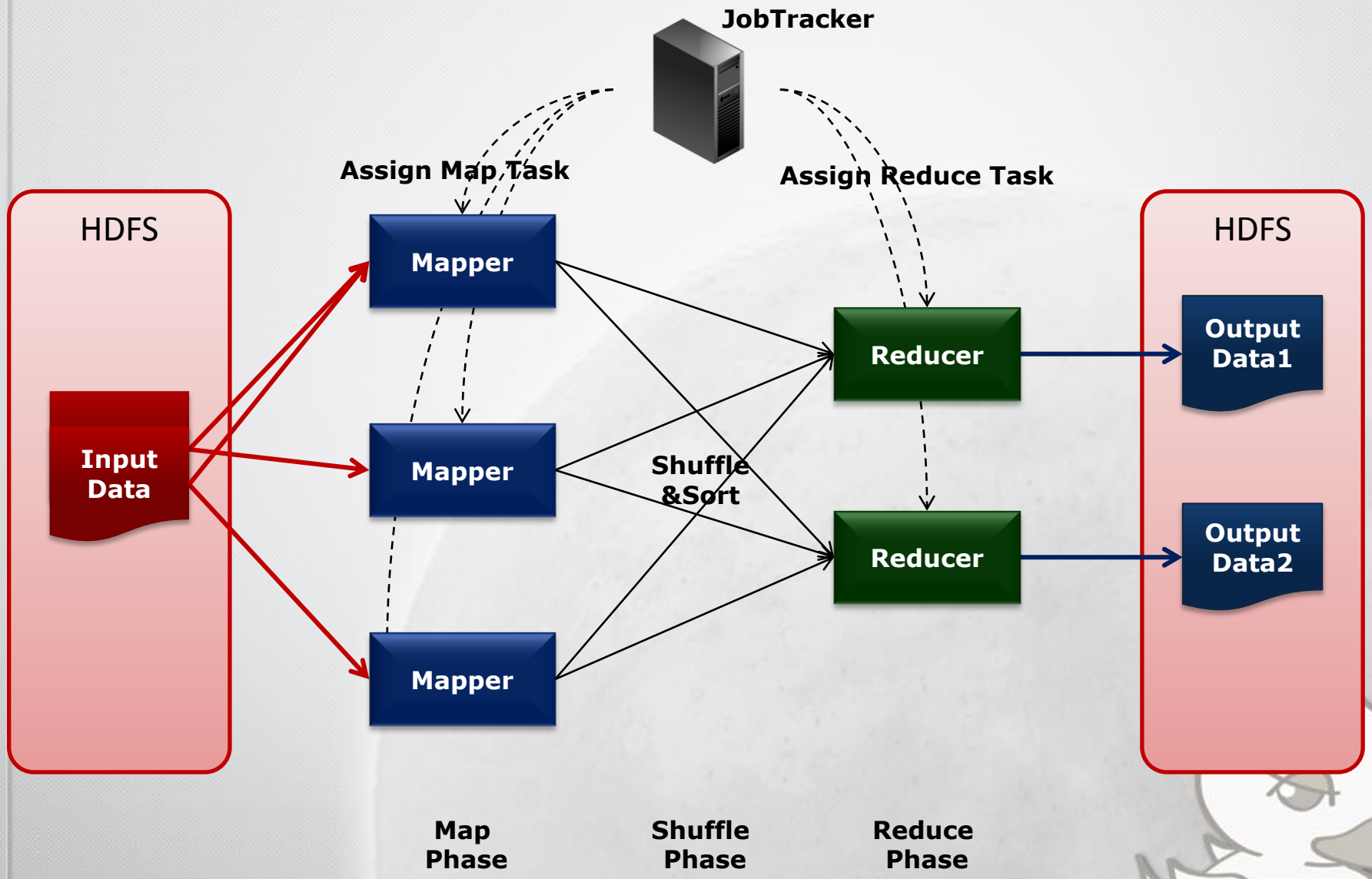
MapReduce ProgrammingModel



MapReduceの各フェーズ

- **Mapフェーズ**
  - map:<key,value> => <key',value'>*
  - 入力１行ごとにmapプログラムを実行する。
- **Combineフェーズ**
  - Mapフェーズの出力結果に対して、Reducerに転送する前に中間処理を施す任意のフェーズ
- **Shuffleフェーズ**
  - shuffle:<key',reducers> => destination reducer
  - Mapフェーズ（Combineフェーズ）の出力結果をReducerの数だけパーティション分割し、同一パーティション内でソートして、Reducerにデータを渡す。
- **Reduceフェーズ**
  - reduce:<key',<value'>*> => <key'',value''>*
  - Shuffleフェーズにて、同じキーのデータが同じReducerに振り分けられる。
  - キーの種類ごとにreduceプログラムを実行する。

HadoopでMapReduce



MapReduce Example
WordCount

MapReduce Example:WordCount

- **Mapper**

```
map(string key, string value) {
    foreach word in value:
        emit(word, 1);
}
```

- **Reducer**

```
reduce(string key, vector<int> values){
    int result = 0;
    for(int i = 0; i < values.size(); i++){
        result += values[i];
    }
    emit(key, result);
}
```

MapReduce Example:WordCount

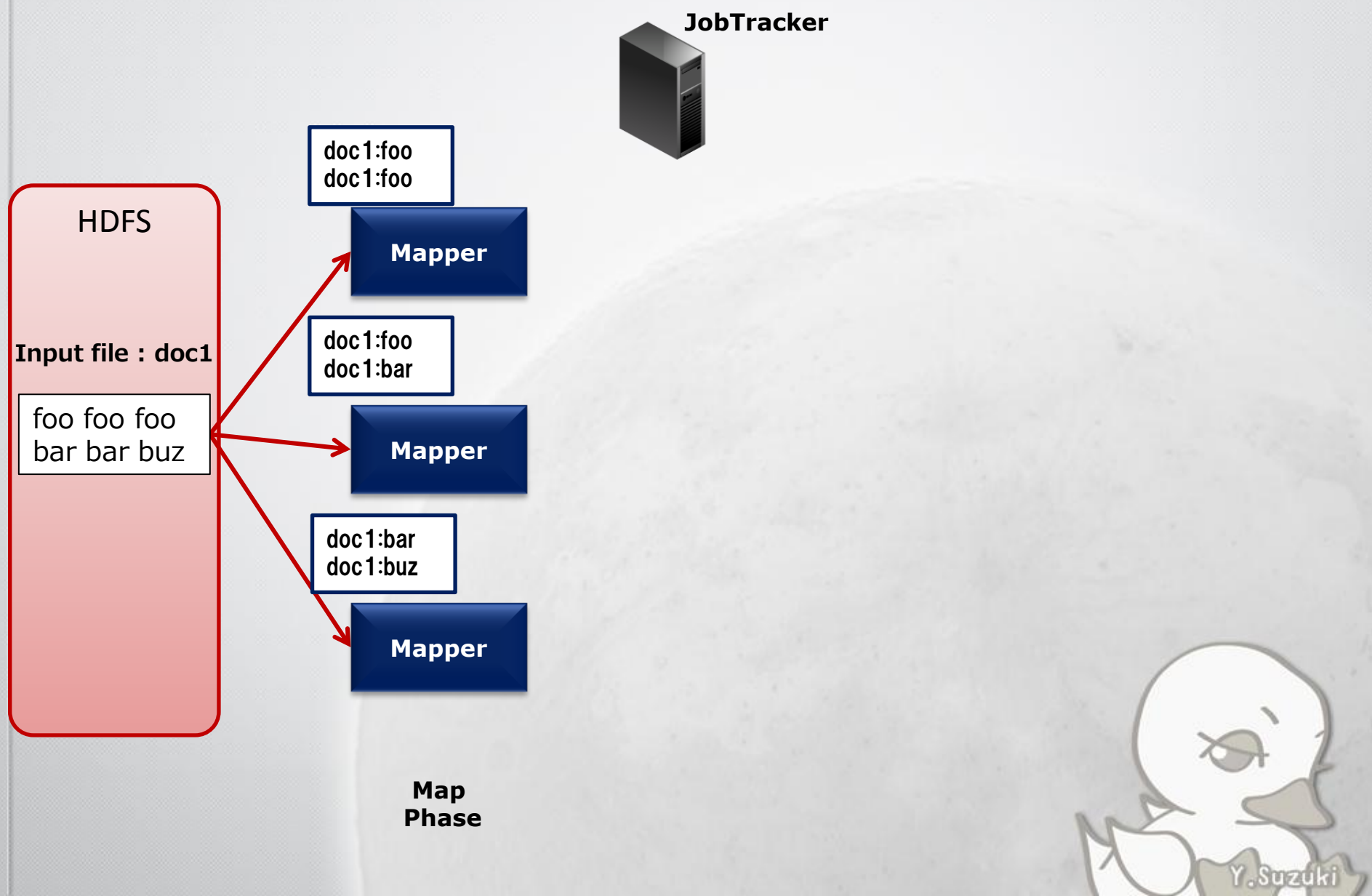
JobTracker

HDFS

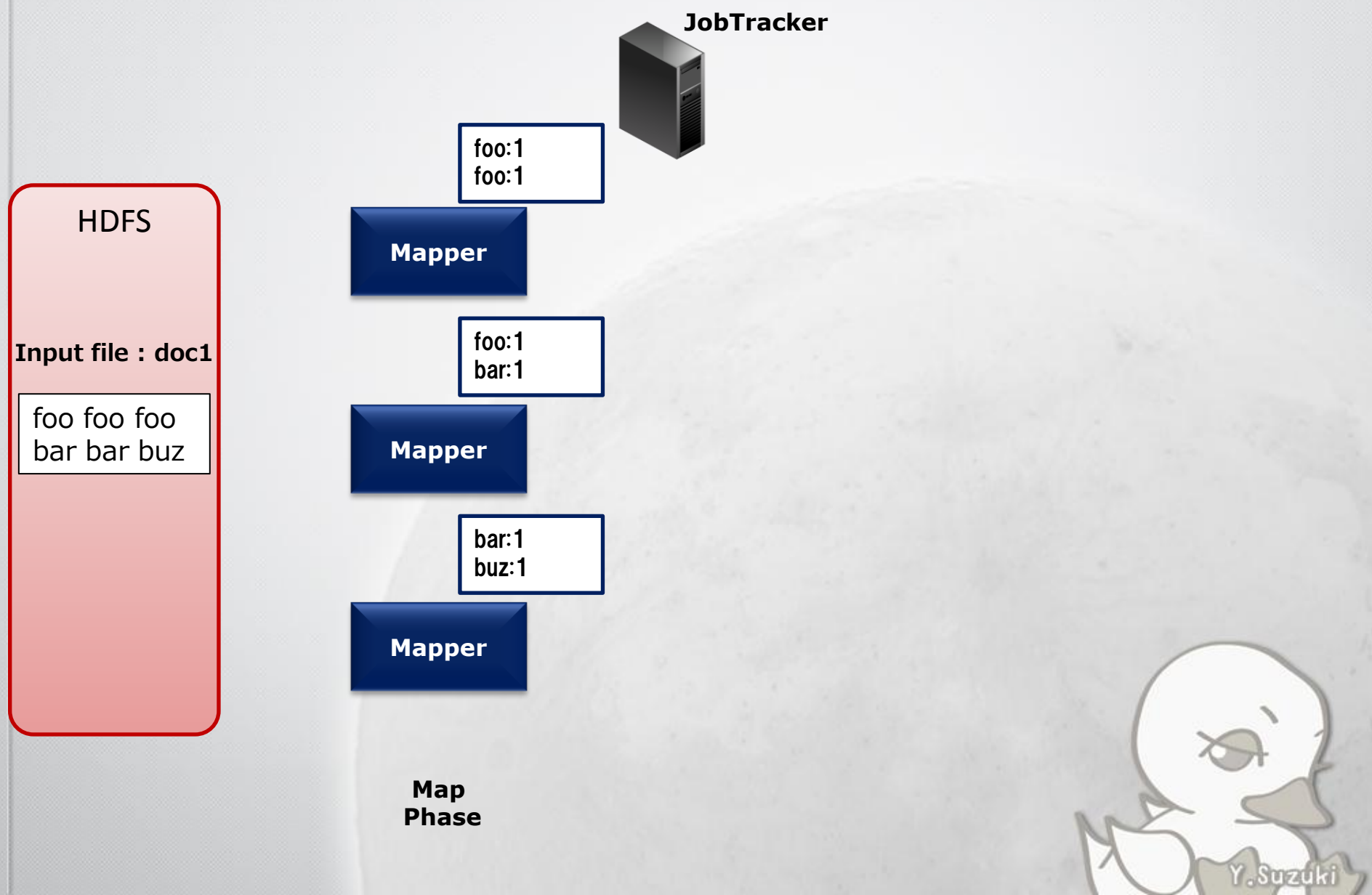
Input file : doc1

foo foo foo
bar bar buz

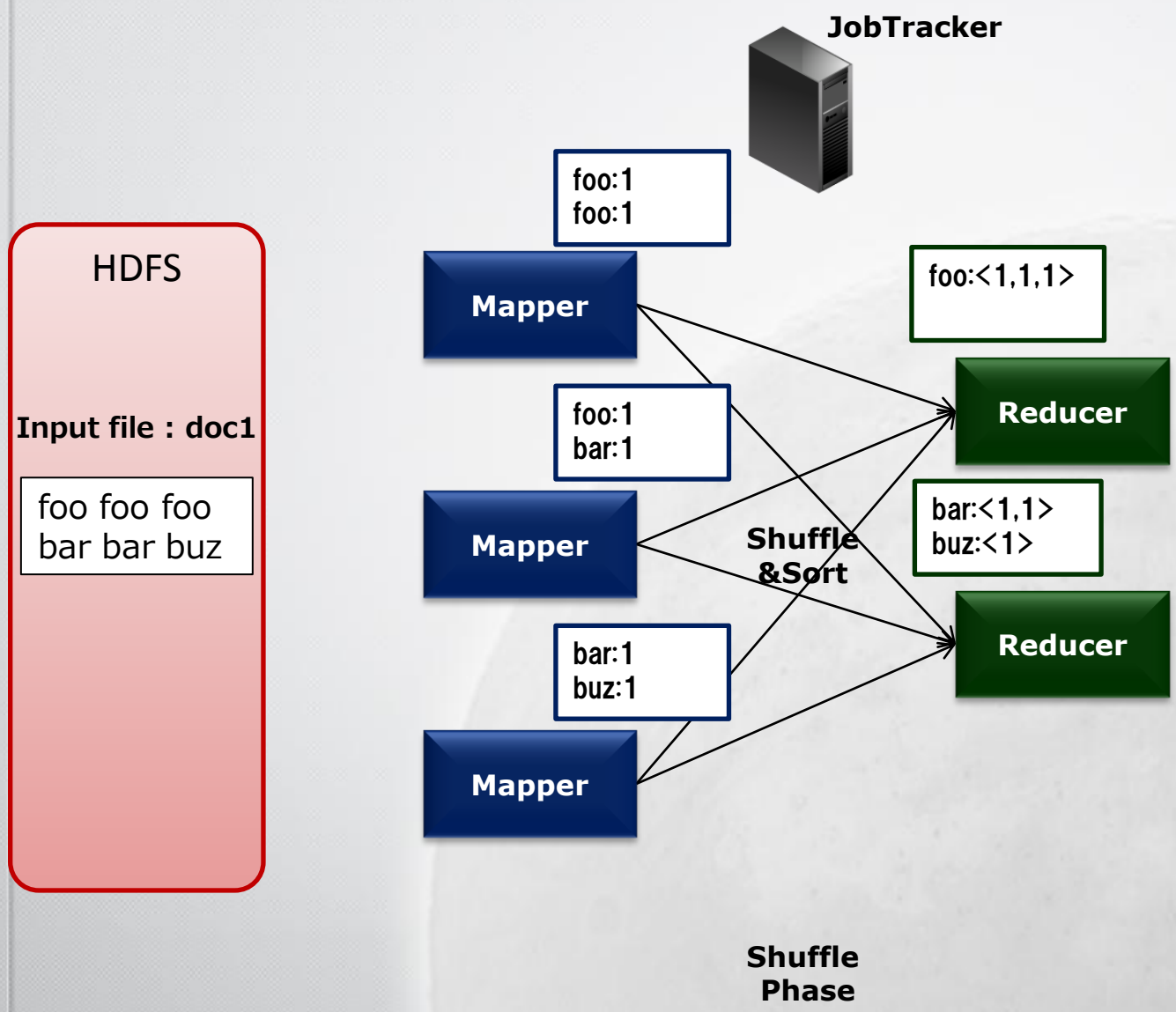
MapReduce Example:WordCount



MapReduce Example:WordCount



MapReduce Example:WordCount



MapReduce Example:WordCount

HDFS

Input file : doc1

foo foo foo
bar bar buz

Mapper

Mapper

Mapper

foo:3

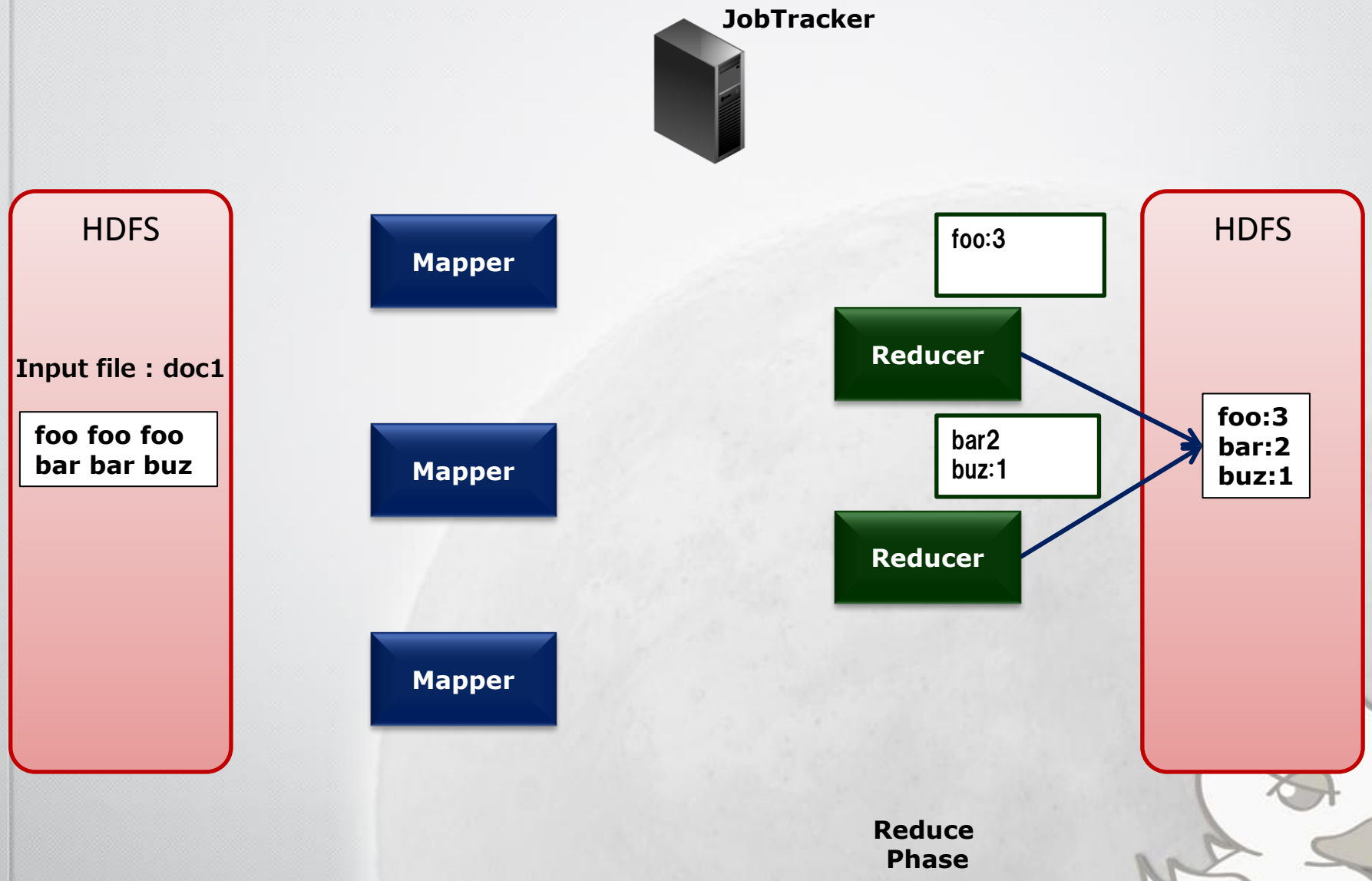
Reducer

bar2
buz:1

Reducer

Reduce Phase

MapReduce Example:WordCount



Hadoop で MapReduce

public static void main ...

Java以外の言語じゃ利用できないの？

そんなことは

ありません

Hadoop で MapReduce

複数の方法でMapReduceを利用することができます。

- **Hadoop Streaming**
  - Java以外の（標準入出力に対応する）言語で記述可能
- **Pig**
  - Yahoo!が開発したPigLatinと呼ばれる処理言語
  - データフローに沿った形で処理内容を定義し、定義した内容をMapReduceに変換して実行できる。

http://pig.apache.org/

- **Hive**
  - Facebookが開発したHiveQLと呼ばれるSQLライクな言語
  - SELECT文を発行すると、MapReduceのタスクが起動。

http://hive.apache.org/

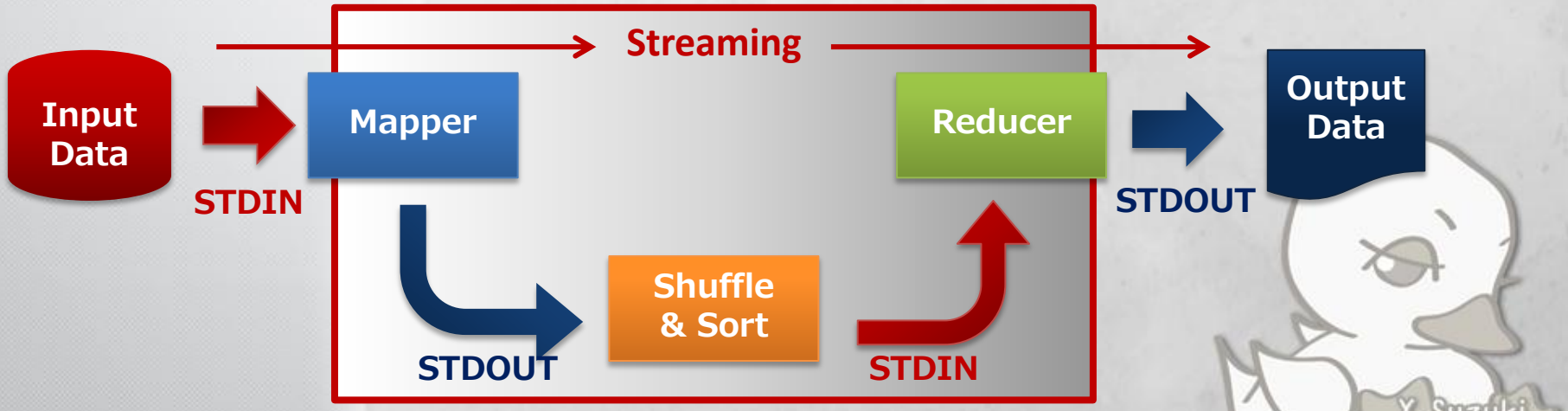
Agenda

1. Hadoopおさらい
2. HadoopStreaming
3. Hive
4. Demo （Apacheログ解析）
5. まとめ

Hadoop Streaming

- **標準入出力の扱える言語で記述可能**
  - Ruby, Perl, PHP, Python, AWK etc…
    使い慣れた言語が活用できる！

- **HadoopStreamingの仕組み**
  - Hadoopの3つの処理フロー（Map・Shuffle&Sort・Reduce）のうち、MapとReduceのフェーズで標準入出力を介して、自由にプログラムを記述できる。



Hadoop Streaming

```
hadoop jar $HADOOP_HOME/contrib/streaming/hadoop-streaming.jar
    -input   inputData
    -output  outputData
    -mapper  map_exec
    -reducer reduce_exec
```

- **-input**
  - 入力ファイルの指定。HDFS上のファイルを指定する。
  - フォルダ名を入れると、フォルダ内全体が対象になる。
- **-output**
  - 出力ディレクトリの指定。HDFS上のディレクトリを指定する。
  - すでに存在しているフォルダ名を指定するとエラーになるので注意。
- **-mapper**
  - MapReduce処理のMapper部分のプログラムを指定する。
  - 単純なコマンドライン実行も記述可能
- **-reducer**
  - MapReduce処理のReducer部分のプログラムを指定する。

Rubyで WordCount

mapper.rb

```
#!/usr/bin/ruby

ARGF.each do |line|
  line.chomp!
  line.split.each do |word|
    puts "#{word}¥t1"
  end
end
```

reducer.rb

```
#!/usr/bin/ruby

data = Hash.new {|h,k| h[k] = 0}
ARGF.each do |line|
  line.chomp!
  key, value = line.split(/¥t/)
  data[key] += 1
end

data.each do |k,v|
  puts "#{k}¥t#{v}"
end
```

InputData（入力データの準備）

- **InputDataは以下のようなかんじで。**

  Change Energy jobs War Energy Economy Change Jobs

  War Jobs Economy Change Jobs Change Jobs Change

  Economy Change Jobs Change Energy jobs War Jobs

- **HDFSにInputDataを転送**

  $ hadoop dfs -copyFromLocal word word

実行

```
$ hadoop jar /usr/lib/hadoop/contrib/streaming/hadoop-streaming-0.20.2-cdh3u3.jar ¥
-input word -output result ¥
-mapper "ruby mapper.rb" -reducer "ruby reducer.rb"
```

途中経過

```
packageJobJar: [/var/lib/hadoop-0.20/cache/cloudera/hadoop-unjar3991896476197815702/] []
/tmp/streamjob1629587214310158736.jar tmpDir=null
12/05/22 13:11:43 WARN snappy.LoadSnappy: Snappy native library is available
12/05/22 13:11:43 INFO util.NativeCodeLoader: Loaded the native-hadoop library
12/05/22 13:11:43 INFO snappy.LoadSnappy: Snappy native library loaded
12/05/22 13:11:43 INFO mapred.FileInputFormat: Total input paths to process : 1
12/05/22 13:11:46 INFO streaming.StreamJob: getLocalDirs(): [/var/lib/hadoop-0.20/cache/cloudera/mapred/local]
12/05/22 13:11:46 INFO streaming.StreamJob: Running job: job_201205221026_0002
12/05/22 13:11:46 INFO streaming.StreamJob: To kill this job, run:
12/05/22 13:11:46 INFO streaming.StreamJob: /usr/lib/hadoop-0.20/bin/hadoop job  -Dmapred.job.tracker=0.0.0.0:8021 -kill
job_201205221026_0002
12/05/22 13:11:46 INFO streaming.StreamJob: Tracking URL: http://0.0.0.0:50030/jobdetails.jsp?jobid=job_201205221026_0002
12/05/22 13:11:47 INFO streaming.StreamJob:  map 0%  reduce 0%
12/05/22 13:13:50 INFO streaming.StreamJob:  map 50%  reduce 0%
12/05/22 13:14:11 INFO streaming.StreamJob:  map 100%  reduce 0%
12/05/22 13:15:28 INFO streaming.StreamJob:  map 100%  reduce 100%
12/05/22 13:15:28 INFO streaming.StreamJob: To kill this job, run:
```

途中経過

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	2	0	0	2	0	1 / 0
reduce	100.00%	1	0	0	0	1	4 / 0

	Counter	Map	Reduce	Total
Job Counters	SLOTS_MILLIS_MAPS	0	0	207,410
	Launched reduce tasks	0	0	4
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0
	Total time spent by all maps waiting after reserving slots (ms)	0	0	0
	Launched map tasks	0	0	3
	Data-local map tasks	0	0	3
	Failed reduce tasks	0	0	1
	SLOTS_MILLIS_REDUCES	0	0	89,240
FileSystemCounters	HDFS_BYTES_READ	395	0	395
	FILE_BYTES_WRITTEN	115,984	0	115,984
	Combine output records	0	0	0
	Map input records	4	0	4
	Physical memory (bytes) snapshot	377,798,656	0	377,798,656

OutputData（出力結果の取得）

- **HDFS上の出力結果の中身をコマンドで取得**

```
$ hadoop dfs -getmerge result result.txt
jobs    2
Energy  3
Change  7
Economy 3
War     3
Jobs    6
```

Hadoop Streaming　まとめ

- Hadoop Streaming なら 標準出力を扱える言語なら なんでも OK
- Map Reduceの利用範囲は広い

Agenda

1. Hadoopおさらい
2. HadoopStreaming
3. Hive
4. Demo （Apacheログ解析）
5. まとめ

Hive

MapReduceのSQLラッパ

- Facebookが開発してHadoopプロジェクトに寄贈
- Hadoopベースのデータ・ウエアハウス（DWH）
- HiveQLというSQLのような言語でHDFS上のデータを操作可能
- HiveQLの実行でSELECT文を発行したら、Map Reduceのタスクが起動

```
# hive -e 'select count(distinct user_id) from logs;
Hive history file =/tmp/root/hive_job_log_root_2xxxxxxx_xxx.txt
Total MapReduce jobs = 1
...
Tracking URL = *********
 map = 0%, reduce = 0%
 map = 5%, reduce = 0%
...
```

メリット

Map.rb

```
#!/usr/bin/ruby

ARGF.each do |line|
  line.chomp!
  line.split.each do |word|
    puts "#{word}¥t1"
  end
end
```

Reducer.rb

```
#!/usr/bin/ruby

data = Hash.new {|h,k| h[k] = 0}
ARGF.each do |line|
  line.chomp!
  key, value = line.split(/¥t/)
  data[key] += 1
end

data.each do |k,v|
  puts "#{k}¥t#{v}"
end
```

select count(distinct user_id) from logs;

MapReduceを意識することなく、SQLライクに記述可能

デメリット

- **MapReduceのオーバヘッド**
  - 内部的にはMapReduceタスクに変換して実行される。
    よってMapReduce起動のオーバヘッド時間は変わらない。
- **行単位のデータ更新ができない。**
  - UPDATE文やDELETE文がない
  - INSERT文は追記(INTO)ではなく、上書き(OVERWRITE)
    ※Hive0.8からはINTOもできるように。
- **HiveQL ≠ SQL標準準拠**
  - 細かい文法的な違いが多々ある。

HiveQL

HiveQL

- **DDL**
  - テーブル作成 (CREATE)
  - テーブル操作 (DROP, ALTER)
  - テーブル情報表示
- **DML**
  - データの格納
  - データの抽出
  - データの絞込み
  - データの並び替え
  - テーブルの結合

テーブル作成

```
CREATE [EXTERNAL] TABLE [IF NOT EXISTS] テーブル名
[(項目名 型 [COMMENT コメント], …)]
[COMMENT テーブルのコメント]
[PARTITIONED BY (項目名 型 [COMMENT コメント], …)]
[CLUSTERED BY (項目名, …) [STORED BY (項目名 [ASC|DESC],
…)] INTO バケット数 BUCKETS]
[ROW FORMAT 行フォーマット]
[
   STORED AS ファイルフォーマット]
 | STORED BY 'クラス名' [WITH SERDEPROPERTIES (～)]
]
[LOCATION パス]
[TBLPROPERTIES (プロパティー名=値, …)]
[AS SELECT文]
```

テーブル作成

指定できるデータ型

型の種類		データ型名	内容
プリミティブ型	整数型	TINYINT	1Byteの整数
		SMALLINT	2Byteの整数
		INT	4Byteの整数
		BIGINT	8Byteの整数
	浮動小数点型	FLOAT	単精度の浮動小数点
		DOUBLE	倍精度の浮動小数点
	論理型	BOOLEAN	TRUEとFALSE
	文字列型	STRING	文字列
配列型		ARRAY < データ型 >	データ型を配列として扱う
マップ型		MAP <プリミティブ型, データ型>	Key, Value形式でデータを扱う
構造型		STRUCT<カラム名：データ型 ,,,>	複数のデータ型を構造化する

テーブル作成

行フォーマット

ROW FORMAT句で行フォーマットを指定し、
ファイルを読み込む時に、ここで指定したフォーマットに従って項目値を分割し、テーブルに格納する。

```
ROW FORMAT DELIMITED
[ FIELDS TERMINATED BY ‘,’]
[ COLLECTION ITEMS TERMINATED BY ‘:’]
[ MAP KEYS TERMINATED BY ‘=’]
[ LINES TERMINATED BY ‘¥n’]
```

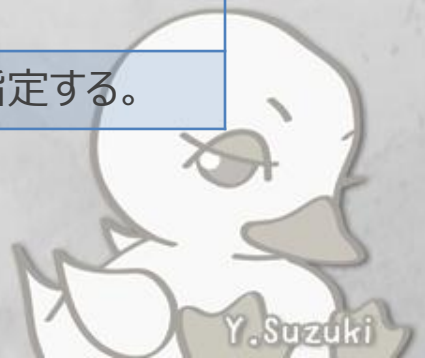
```
ROW FORMAT SERDE 'SerDeクラス名' [WITH
SERDEPROPERTIES(プロパティー名=値, …)]
```

テーブル作成

行フォーマット

ROW FORMAT句で行フォーマットを指定し、
ファイルを読み込む時に、ここで指定したフォーマットに従って項目値を分割し、テーブルに格納する。

構文	内容
FIELDS	フィールド（項目）の区切り文字を指定する。 デフォルトは0x01。
COLLECTION ITEMS	コレクション（配列）の区切り文字を指定する。 デフォルトは区切り文字無し。
MAP KEYS	マップのキーと値の区切り文字を指定する。 デフォルトは区切り文字無し。
LINES	行の区切り文字を指定する。 デフォルトは¥n
SERDE	SerDe（シリアライズ・デシリアライズ）クラスを指定する。



テーブル作成

サンプル１

TSVファイルをタブで区切った値として1行ごとに格納する。

```
CREATE TABLE logs(id INT, created_at STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘,‘
LINES TERMINATED BY'¥n';
```

サンプル２

date項目でPartisionを設定する。
(Partitionを設定するとそのカラムでの項目ごとの集計が簡単にできる。)

```
CREATE TABLE logs(id INT, created_at STRING)
PARTITIONED BY(date STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ‘,‘
LINES TERMINATED BY'¥n';
```

テーブル操作

- テーブル削除
  - DROP TABLE [IF EXISTS] テーブル名;
    - dropするとデータファイル自体が消える。
    - CREATE EXTERNALで作成されたテーブルは、削除されずそのまま残る。
- テーブル名変更
  - ALTER TABLE テーブル名　RENAME TO 新テーブル名;

テーブル情報表示

- テーブルの一覧を表示
  - SHOW TABLES;
- パテーション表示
  - SHOW PARTITIONS テーブル名;
- テーブルの内容を表示
  - DESCRIBE [EXTENDED] テーブル名;
    - 省略形のDESCも使用可能
    - EXTENDEDを付けると詳細情報を表示

データの格納

LOAD文

- LOAD DATA [LOCAL] INPAT ‘パス文字列’
  [OVERWRITE] INTO TABLE テーブル名
  [PARTITION (項目名=値, …)]

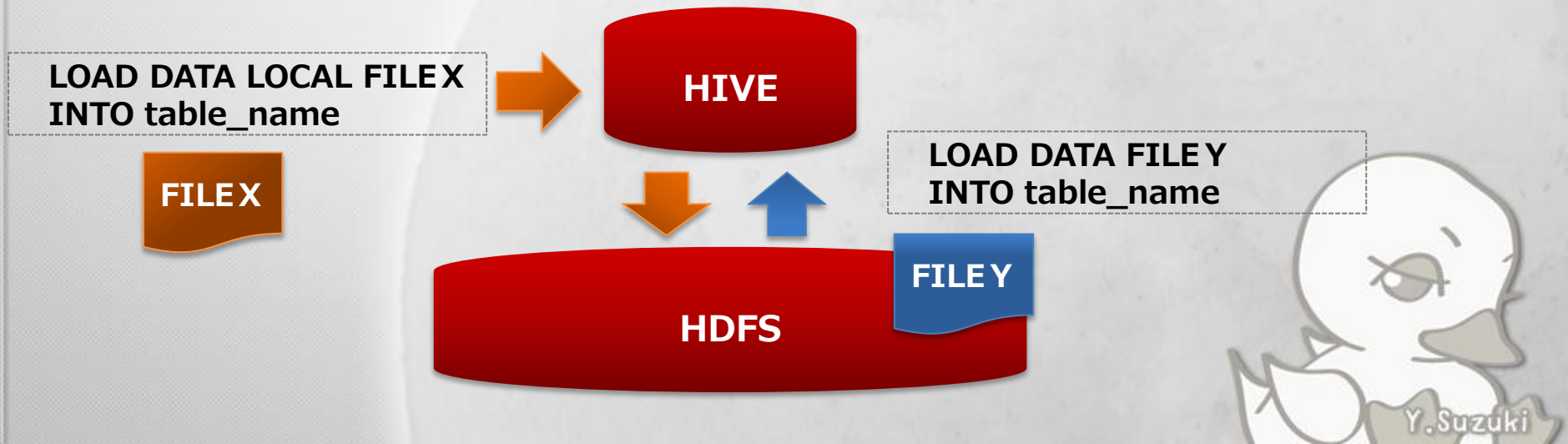
- Hive外からのデータの取り込み場合に使用し、
  HDFS上、ローカルディスク上からそれぞれデータを格納することができます。

- OVERWRITEを付けた場合、上書き(新しいデータのみ)
  付けない場合は、データ追加

- パーテションを利用しているテーブルの場合、特定のパーテションを指定してデータを格納することができる。

データの格納

- LOAD文
  - HDFS上のファイル
    - Hive領域にmvコマンドのイメージ
      HDFS上の元ファイルは消えてしまう。
  - ローカルディスク上のファイル
    - Hive領域にputコマンドのイメージ



データの格納

INSERT OVERWRITE文

- INSERT OVERWRITE TABLE テーブル名
  [PARTITION (項目名=値, …)]
  SELECT文 FROM 元テーブル名

- テーブル（SELECT文）から別のテーブルに対し、
  データを入れるのに使用する。
  この際テーブルのデータはすべて上書きされる。

- HDFS上やローカルディスク上にもファイルとして保存可能。
  INSERT OVERWRITE [LOCAL] DIRECTORY ‘パス文字列’
  SELECT文 FROM ~

データの抽出

- SELECT文（基本構文）
  - SELECT [ALL | DISTINCT] カラム名1,カラム名2, ...
    FROM テーブル名
    [WHERE 条件]
    [GROUP BY col_list]
    [CLUSTER BY col_list | [DISTRIBUTE BY col_list]
    [SORT BY col_list]]
    [LIMIT number]
  - from テーブル名 select ～ where 条件;
    - FROM句を先頭に持ってくることもできる。
      SELECTが先のクエリと等価。
  - テーブル名は2個以上指定不可(結合はJOIN句を使う)

データの絞込み

- **WHERE句(条件指定)**
  - SELECT * FROM テーブル名 WHERE 条件;
- **DISTINCT(重複削除)**
  - SELECT DISTINCT カラム名 FROMテーブル名;
- **副問い合わせ**
  - SELECT カラム名1,カラム名2, ...
    FROM (
    　SELECT カラム名1,カラム名 FROM テーブル名
) テーブル別名;

データの並び替え

ORDER BY（全出力結果のソート）

- SELECT カラム名1，カラム名2 FROM テーブル名
  ORDER BY カラム名 （ASC | DESC）

- ORDER BYは全データソートする必要があるため、
  Reduce処理を1つに集約する。よって低速。

SORT BY（Reducerごとのソート）

- SELECT カラム名1，カラム名2 FROM テーブル名
  SORT BY カラム名 （ASC | DESC）

- HadoopのShuffleフェーズ相当

データの集計処理

- **GROUP BY句(グループ化)**
  - SELECT * FROM テーブル名 GROUP BY カラム名;
- **HAVING句(グループ化後の条件)**
  - SELECT * FROM テーブル名 GROUP BY カラム名;
    HAVING 条件;
- **LIMIT句**
  - SELECT * FROM テーブル名 LIMIT Number;

  以下のようなクエリは、MapReduceしないので高速

  ```
  SELECT * FROM table_name LIMIT 3
  ```

テーブルの結合

- **JOIN句**
  - SELECT * FROM テーブル名１ t1 JOIN テーブル名2 t2 ON t1.c1 = t2.c1;
  - FROM句でのテーブル名の羅列不可
    → 結合する場合は、JOIN句を使用する。
  - ~~SELECT t1.c1, t2.c2 FROM t1, t2 WHERE t1.c1 = t2.c1;~~
  - SELECT t1.c1, t2.c2 FROM t1 JOIN t2 ON t1.c1 = t2.c1;

Hive

- プログラミング言語でMapReduceを書く必要がなく、RDBのようにMapReduceをQLで操作できるのは魅力。
- MapReduceした結果をHDFSやローカルファイルに落とす等比較的柔軟に利用ができる。

Agenda

1. Hadoopおさらい
2. HadoopStreaming
3. Hive
4. Demo （Apacheログ解析）
5. まとめ

Apacheログ解析

ApacheのCommonログ（実データ）を使って、IPアドレス順に件数を集計してみる。


```
googlebot.com - - [01/May/2012:00:07:57 +0900] "GET /robots.txt HTTP/1.1" 404 216 "-" "Mozilla/5.0 (compatible; Googlebot/2.1;
+http://www.google.com/bot.html)"
rate-limited-proxy-209-85-238-7.google.com - - [01/May/2012:00:07:59 +0900] "GET / HTTP/1.1" 200 21120 "-" "DoCoMo/2.0
N905i(c100;TB;W24H16) (compatible; Googlebot-Mobile/2.1; +http://www.google.com/bot.html)"
rate-limited-proxy-209-85-238-7.google.com - - [01/May/2012:00:28:53 +0900] "GET /?feed=rss2 HTTP/1.1" 304 0 "-" "Feedfetcher-Google;
(+http://www.google.com/feedfetcher.html; 2 subscribers; feed-id=16492007851965158684)"
niederhausen.tm.informatik.uni-leipzig.de - - [01/May/2012:00:33:56 +0900] "GET /robots.txt HTTP/1.1" 404 216 "-" "findlinks/2.5
(+http://wortschatz.uni-leipzig.de/findlinks/)"
61-23-123-166.rev.home.ne.jp - - [01/May/2012:01:06:33 +0900] "GET /wp-content/themes/inove_labsOT/style.css HTTP/1.1" 200 29755
"http://labs.opentone.co.jp/" "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)"
61-23-123-166.rev.home.ne.jp - - [01/May/2012:01:06:33 +0900] "GET /wp-content/themes/inove_labsOT/style.css HTTP/1.1" 200 29755
"http://labs.opentone.co.jp/" "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0)"
61-23-123-166.rev.home.ne.jp - - [01/May/2012:01:06:33 +0900] "GET /wp-content/themes/inove_labsOT/pagenavi-css.css?ver=2.60
HTTP/1.1"
```

Apacheログ解析：テーブル作成

```
CREATE TABLE apache_log (
    ip_address STRING, identd STRING, user STRING,
    finish_time STRING, request_line string,
    return_code INT, size INT)
ROW FORMAT SERDE
'org.apache.hadoop.hive.serde2.dynamic_type.DynamicSerDe'
WITH SERDEPROPERTIES (
'serialization.format'='org.apache.hadoop.hive.serde2.thr
ift.TCTLSeparatedProtocol',
'quote.delim'='("|¥¥[|¥¥])',
'field.delim'=' ',
'serialization.null.format'='-')
STORED AS TEXTFILE;
```

Apacheログ解析：データ格納

- アパッチのログファイルをテーブルに格納します。

```
LOAD DATA LOCAL INPATH
'/tmp/20120524/demo/access_log‘ OVERWRITE INTO TABLE
apache_log;
```

Apacheログ解析：集計

- 準備が整ったら下記クエリを実行します。

```
SELECT ip_address, COUNT(1) AS num
FROM apache_log GROUP BY ip_address
SORT BY num DESC;
```

Apacheログ解析：結果

```
crawl-66-249-69-58.googlebot.com        21
p2199-ipbf1409funabasi.chiba.ocn.ne.jp  21
76-237-95-3.lightspeed.rcsntx.sbcglobal.net     7
rate-limited-proxy-209-85-238-117.google.com    3
crawl-119-235-237-16.naver.jp   2
122.208.75.82   2
126.210.117.174 2
157.55.116.13   2
219.96.12.198   2
64.246.165.170  2
66.249.69.137   2
```

Agenda

1. Hadoopおさらい
2. HadoopStreaming
3. Hive
4. Demo （Apacheログ解析）
5. まとめ

まとめ

- HadoopはJava専用と思われがちですが、
  様々なユーテリティ群を活用することで、
  Java以外の言語でも使いこなすことができます。

- 既存システムを活かしつつ、本当に必要な箇所だけ分散処理によりスケールアウトをしていくことで、
  高価なシステムを導入したり構築しなくても、大量データの活用が実現可能になります。

まとめ

普段使っている開発言語で、
大量の未活用のデータをHadoopで、
有効活用してみましょう！

ご清聴ありがとうございました